## Методы создания алгоритмически безопасных процедур.

###### 2.4.1. Постановка задачи

Основной отличительной особенностью подхода, связанного с созданием алгоритмически безопасного ПО является то, что начало процесса обеспечения безопасности программ при их разработке можно перенести на более ранние этапы жизненного цикла программного обеспечения, например, на этапы, предшествующие этапу испытания программ, тем самым увеличив общее время на внесение в программы защитных функций. Здесь уместно процитировать слова Э.В. Дейкстра, одного из основоположников современной методологии программирования, сказанные им еще в 1972 году: "В настоящее время общепринятой техникой является составление программы, а затем ее тестирование. Однако тестирование программ может быть очень эффективным способом демонстрации наличия ошибок, но оно безнадежно неадекватно для доказательства их отсутствия... Не следует сначала писать программу, а потом доказывать ее правильность, поскольку в этом случае требование найти доказательство только увеличит тяготы бедного программиста". Эти слова как нельзя лучше подходят и к современной проблематике, связанной, в данном случае, не столько с разработкой правильных, сколько безопасных программ. Иными словами, ключом к созданию безопасного программного обеспечения является стремление защищаться от дефектов с самого начала жизненного цикла программ, а не после факта их создания.

Кроме того, следует отметить, что одно из основных достоинств методов создания алгоритмически безопасного программного обеспечения заключается в том, что данные методы позволяют защищать программное обеспечение, как на этапе разработки, так и на этапе его эксплуатации.

В то же время, необходимо сказать, что основная сложность в разработке безопасных программ указанного типа заключается в трудности нахождения эффективных алгоритмов их функционирования, которые являлись к тому же доказуемо корректными. И, предположительно, проблема неразрешимости доказательства безопасности для сложных программ при использовании методов их тестирования на этапе испытаний остается, в случае применения методов создания алгоритмически безопасного программного обеспечения, по-прежнему неразрешимой, что будет видно из дальнейших рассуждений.

Одним из главных методических вопросов создания безопасного программного обеспечения является постановка задачи (формулировка проблемы) ее разработки. Предположим, некоторому разработчику (коллективу разработчиков) предписывается разработать программу P для некоторого объекта автоматизации. При этом последствия некорректного функционирования программы таковы, что могут привести к неким "нежелательным" или даже катастрофическим последствиям для объекта автоматизации. Исходя из гипотезы, что в общем случае проблема доказательства безопасности для сложных комплексов программ является неразрешимой, неформальная качественная постановка задачи разработки указанных программ может быть тогда сформулирована следующим образом.

Постановка 1. "При некоторых условиях и ограничениях необходимо разработать программу P, которая корректно вычисляет результат почти для всех своих входных значений".

Таким образом, разработчик констатирует факт, что лишь для незначительной (возможно достаточно малой) доли входных значений программа может выдать некорректное выходное значение в результате необнаруженного программного дефекта или активизации необнаруженной программной закладки.

Понятно, что в некоторых случаях возникает определенная необходимость получения каких-либо более точных количественных оценок степени защиты созданной программы от внутренних дефектов, с целью получения определенных гарантий надежного функционирования объекта автоматизации.

При использовании контрольно-испытательных методов анализа безопасности программ, когда анализу подвергается только исполняемый код программы можно получить (как правило, экспертным путем) либо приближенные количественные характеристики обнаружения дефектов в контролируемых программах, либо стратификационные характеристики не нарушения программой некоторых условий безопасности. В этом случае постановка 1 задачи разработки программы P принципиально не меняется. (Естественно не рассматриваются простейшие случаи, когда при тестировании возможен контроль результата программы при полном переборе всех входных значений, при всех ограничениях и допущениях и т.п.).

Постановка же задачи разработки безопасного программного обеспечения за счет алгоритмически безопасных процедур меняется с точки зрения получения точных количественных (в данном случае вероятностных) характеристик кардинальным образом.

Постановка 2. "При некоторых условиях и ограничениях необходимо разработать программу P\*, которая корректно вычисляет результат для всех своих входных значений с пренебрежимо малой вероятностью ошибки". На примере постановок 3, 4 и 5, 6 можно также проследить существенную разницу в парадигме разработки безопасных программ. Постановки 3 и 4 в отличие от постановок 1 и 2 учитывают не "содержание" наборов входных значений программы P, а их распределения вероятностей, а постановки 5 и 6 являются в некотором смысле обобщенными и учитывают заданный структурный критерий тестирования.

Постановка 3. "При некоторых условиях и ограничениях необходимо разработать программу P, которая некорректно вычисляет результат лишь для некоторого частного распределения вероятностей своих входных величин".

Постановка 4. "При некоторых условиях и ограничениях необходимо разработать программу P\*, которая корректно вычисляет результат для любого распределения вероятностей своих входных величин с пренебрежимо малой вероятностью ошибки".

Постановка 5."При некоторых условиях и ограничениях необходимо разработать программу P, которая корректно вычисляет результат почти на всех тестах полной системы тестов программы относительно заданного структурного критерия тестирования".

Постановка 6. "При некоторых условиях и ограничениях необходимо разработать программу P\*, которая корректно с пренебрежимо малой вероятностью ошибки вычисляет результат на всех тестах полной системы тестов относительно заданного структурного критерия тестирования".

Одним из принципиальных условий решения задачи в постановке 2, 4 и 6 является наличие некоего свойства алгоритмической трансформации, позволяющего переходить от традиционного пути создания программ, которые будут затем проверяться на наличие дефектов, к априорно безопасным программам. К числу таких примеров можно отнести самокорректирующиеся и самотестирующиеся программы, обладающие свойством случайной самосводимости и программы, созданные на базе методов конфиденциальных вычислений, начало изучения которых было положено в работах. Рассмотренные до этого методы анализа безопасности ПО связаны с попытками обезопасить фактически уже разработанное программное обеспечение от действий злоумышленника. Это означает, что разработка безопасного ПО возможна за счет создания средств противодействия программным закладкам для продуктов, созданных на основе существующих информационных технологий создания программного обеспечения. То есть, только после факта разработки программ начинается верификационный анализ, тестирование или контроль их на технологическую безопасность. В этом смысле проблема обеспечения технологической безопасности программного обеспечения более близка к фундаментальной проблеме его надежности.

В то же время, данные методы создания безопасного ПО характеризуются рядом существенных недостатков, к числу которых можно отнести: невозможность перекрытия для большинства программных комплексов всего спектра тестовых наборов исходных данных; необходимость создания высоконадежных механизмов тестирования программ, например, механизмов экстраполяции результатов испытаний; существенные ресурсозатраты на проведение испытаний и т.п.

Поэтому в последнее время появилась насущная необходимость в создании новых информационных технологий разработки ПО, исходно ориентированных на создание безопасных программных продуктов относительного заданного класса. В этом случае проблема исследований сводится к разработке таких математических моделей, которые представляются адекватной формальной основой для создания методов защиты программного обеспечения на этапе его проектирования и разработки. При этом изначально предполагается, что:

* один или несколько участников проекта являются (или, по крайней мере, могут быть) злоумышленниками;
* в процессе эксплуатации злоумышленник может вносить в программы изменения (необязательно связанные с внедрением апостериорных программных закладок);
* средства вычислительной техники, на которых выполняются программы, не свободны от аппаратных закладок.

Тогда, исходя из этих допущений, формулируется следующая неформальная постановка задачи: "Требуется разработать программное обеспечение таким образом, что несмотря на указанные выше "помехи" оно функционировало бы правильно". Одно из основных достоинств здесь состоит в том, что одни и те же методы позволяют защищаться от злоумышленника, действующего как на этапе разработки, так и на этапе эксплуатации программного обеспечения. Однако, это достигается за счет некоторого замедления вычислений, а также повышения затрат на разработку программного обеспечения.

В рамках указанного выше подхода на данный момент известны два направления, неформальное введение в которые дается ниже.

###### 2.4.2. Методы создания самотестирующихся и самокорректирующихся программ для решения вычислительных задач

Общие принципы создания двухмодульных вычислительных процедур и методология самотестирования Пусть необходимо написать программу P для вычисления функции fтак, чтобы P(x)=f(x) для всех значений x. Традиционные методы верификационного анализа и тестирования программ не позволяют убедиться с вероятностью близкой к единице в корректности результата выполнения программы, хотя бы потому, что тестовый набор входных данных, как правило, не перекрывают весь их возможный спектр. Один из методов решения данной проблемы заключается в создании так называемых самокорректирующихся и самотестирующихся программ, которые позволяют оценить вероятность некорректности результата выполнения программы, то есть, что P(x)*http://citforum.ru/security/articles/kazarin/nerav.gif*f(x) и корректно вычислить f(x) для любых x, в том случае, если сама программа P на большинстве наборах своих входных данных (но не всех) работает корректно.

Чтобы добиться корректного результата выполнения программы P, вычисляющей функцию f, нам необходимо написать такую программу Tf, которая позволяла бы оценить вероятность того, что P(x)*http://citforum.ru/security/articles/kazarin/nerav.gif*f(x)для любых x. Такая вероятность будет называться вероятностью ошибки выполнения программы P. При этом Tf может обращаться к P как к своей подпрограмме.

Обязательным условием для программы Tf является ее принципиальное отличие от любой корректной программы вычисления функции f, в том смысле, что время выполнения программы Tf, не учитывающее время вызовов программы P, должно быть значительно меньше, чем время выполнения любой корректной программы для вычисления f. В этом случае, вычисления согласно Tf некоторым количественным образом должны отличаться от вычислений функции f и самотестирующаяся программа может рассматриваться как независимый шаг при верификации программы P, которая предположительно вычисляет функцию f. Кроме того, желательное свойство для Tf должно заключаться в том, чтобы ее код был насколько это возможно более простым, то есть Tf должна быть эффективнойв том смысле, что время выполнения Tf  даже с учетом времени, затраченное на вызовы P должно составлять константный мультипликативный фактор от времени выполнения P. Таким образом, самотестирование должно лишь незначительно замедлять время выполнения программы P.

Самокорректирующаяся программа — это вероятностная программа Cf, которая помогает программе Pскорректировать саму себя, если только P выдает корректный результат с низкой вероятностью ошибки, то есть для любого x, Cf вызывает программу P для корректного вычисления f(x), в то время как собственно сама P обладает низкой вероятностью ошибки.

Самотестирующейся/самокорректирующейся программной парой называется пара программ вида(Tf,Cf). Предположим пользователь может взять любую программу P, которая целенаправленно вычисляетf и тестирует саму себя при помощи программы Tf. Если P проходит такие тесты, тогда по любому x, пользователь может вызвать программу Cf, которая, в свою очередь, вызывает P для корректного вычисления f(x). Даже если программа P, которая вычисляет значение функции f некорректно для некоторой небольшой доли входных значений, ее в данном случае все равно можно уверенно использовать для корректного вычисления f(x) для любого x. Кроме того, если удастся в будущем написать программу P' для вычисления f, тогда некоторая пара (Tf,Cf) может использоваться для самотестирования и самокоррекции P' без какой-либо ее модификации. Таким образом, имеет смысл тратить определенное количество времени для разработки самотестирующейся/ самокорректирующейся программной пары для прикладных вычислительных функций.

Перед тем как перейти к более формальному описанию определений самотестирующихся и самокорректирующихся программ необходимо дать определение вероятностной оракульной программе (по аналогии с вероятностной оракульной машиной Тьюринга).

Вероятностная программа M является вероятностной оракульной программой, если она может вызывать другую программу, которая является исполнимой во время выполнения M. Обозначение MAозначает, чтоM может делать вызовы программы A.

Пусть P- программа, которая предположительно вычисляет функцию f. Пусть I является объединением подмножеств In, где n принадлежит множеству натуральных чисел Nи пусть Dp={Dn|n http://citforum.ru/security/articles/kazarin/e.gif N} есть множество распределений вероятностей Dn над In. Далее, пусть err(P,f,Dn) это вероятность того, что P(x)*http://citforum.ru/security/articles/kazarin/nerav.gif*f(x), где x выбрано случайно в соответствии с распределением Dn из подмножества In. Пусть β - есть некоторый параметр безопасности. Тогда (ε1, ε2)-самотестирующейся программой для функции f в отношении Dp с параметрами 0 http://citforum.ru/security/articles/kazarin/b.gif ε1<ε2 http://citforum.ru/security/articles/kazarin/b.gif 1 - называется вероятностная оракульная программа Tf, которая для параметра безопасности β и любой программы P на входе n имеет следующие свойства:

- если err(P,f,Dn)*http://citforum.ru/security/articles/kazarin/b.gif*ε1, тогда программа TfP выдаст на выходе ответ "норма" с вероятностью не менее1-β .  
- если err(P,f,Dn)*http://citforum.ru/security/articles/kazarin/b.gif*ε2, тогда программа TfP выдаст на выходе "сбой" с вероятностью не менее 1-β .

Оракульная программа Cf с параметром 0 http://citforum.ru/security/articles/kazarin/b.gif ε < 1 называется ε-самокорректирующейся программой для функции f в отношении множества распределений Dp, которая имеет следующее свойство по входу n,x*http://citforum.ru/security/articles/kazarin/e.gif*In и β. Если err(P,f,Dn)*http://citforum.ru/security/articles/kazarin/b.gif*ε , тогда CfP=f(x) с вероятностью не менее 1- β.  
(ε1,ε2,ε)-самотестирующейся/ самокорректирующейся программной парой для функции f называется пара вероятностных программ (Tf,Cf) такая, что существуют константы 0 http://citforum.ru/security/articles/kazarin/b.gif ε1 < ε2 http://citforum.ru/security/articles/kazarin/b.gif ε < 1 и множество распределений Dp при которых Tf -есть (ε1,ε2)-самотестирующаяся программа для функции f в отношенииDp и Cf- есть ε-самокорректирующаяся программа для функции f в отношении распределения Dp.

Свойство случайной самосводимости. Пусть x*http://citforum.ru/security/articles/kazarin/e.gif*In и пусть c > 1 - есть целое число. Свойство случайной самосводимости заключается в том, что существует алгоритм A1, работающий за время пропорциональное nO(1), по-средством которого функция f(x) может быть выражена через вычислимую функцию F от x, a1,...,acи f(a1),...,f(ac) и алгоритм A2, работающий за время пропорциональное nO(1), посредством которого по данному x можно вычислить a1,...,ac, где каждое ai является случайно распределенным над In в со-ответствии с Dp.

Метод создания самотестирующейся расчетной программы с эффективным тестирующим модулем

В качестве расчетной программы рассматривается любая программа, решающая задачу получения значения некоторой вычислимой функции. При этом под верификацией расчетной программы понимается процесс доказательства того, что программа будет получать на некотором входе истинные значения исследуемой функции. Иными словами, верификация расчетной программы направлена на доказательство отсутствия преднамеренных и (или) непреднамеренных программных дефектов в верифицируемой программе.

В данном случае предлагается метод создания самотестирующихся программ для верификации расчетных программных модулей. Данный метод не требует вычисления эталонных значений и является независимым от используемого при написании расчетной программы языка программирования, что существенно повышает оперативность исследования программы и точность оценки вероятности отсутствия в ней программных дефектов. Следует в то же время отметить, что предположительно предлагаемый метод можно использовать для программ, вычисляющих функции особого вида, а именно функции, обладающие свойством случайной само-сводимости.

Пусть для функции Y = f (X) существует пара функций (gc, hc)Y таких, что:

Y = gc (f (a1), ..., f (ac)),

X = hc (a1, ..., ac).

Легко увидеть, что если значения ai выбраны из In в соответствии с распределением Dp, тогда пара функций (gc, hc)Y обеспечивает выполнение для функции Y = f (X) свойства случайной самосводимости. Пару функций (gc, hc)Y будем называть ST-парой функций для функции Y = f (X).

Метод верификации расчетных программ на основе ST-пары функций.

Предположим, что на ST-пару функций можно наложить некоторую совокупность ограничений на сложность программной реализации и время выполнения. В этом случае, пусть длина кода программ, реализующих функции gc и hc , и время их выполнения составляет константный мультипликативный фактор от длины кода и времени выполнения программы P. Предлагаемый метод верификации расчетной программы P на основе ST-пары функций для некоторого входного значения вектора X\* заключается в выполнении следующего алгоритма. (Всюду далее, если осуществляется случайный выбор значений, этот выбор выполняется в соответствии с распределением вероятностей Dp).

Алгоритм***ST***

Определить множество A\*={a1\*,...,ac\*} такое, что X\*=hc{a1\*,...,ac\*} , где a1\*,...,ac\* выбраны случайно из входного подмножества In.  
Вызвать программу P для вычисления значения Y0\*=f(X\*)  
Вызвать c раз программу P для вычисления множества значений {f(a1\*),...,f(ac\*)}  
Определить значения Y1\*=gc{f(a1\*) ,...,f(ac\*)}

Если Y0\*=Y1\* , то принимается решение, что программа P корректна на множестве значений входных параметров {X\*,a1\*,...,ac\*} в противном случае данная программа является некорректной.

Таким образом, данный метод не требует вычисления эталонных значений и за одну итерацию позволяет верифицировать корректность программы P на (n+1) значении входных параметров. При этом время верификации можно оценить как http://citforum.ru/security/articles/kazarin/10.gif где ti и tx - время выполнения программы P при входных значениях ai и X\* соответственно;

tgи th-1 - время определения значения функции gc и множества A\* соответственно:

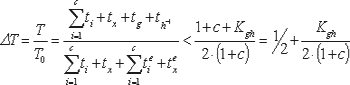
TP(X) - временная (не асимптотическая) сложность выполнения программы P;

Kgh(X, c)- коэффициент временной сложности программной реализации функции gc и определения A\* по отношению ко временной сложности программы P (по предположению он составляет константный мульти-пликативный фактор от TP(X), а его значение меньше 1). Для традиционного вышеуказанного метода тестирования время выполнения и сравнения полученного результата с эталонным значением составляет:

http://citforum.ru/security/articles/kazarin/11.gif

где tieи txe - время определения эталонных значений функции Y = f (X) при значениях aiи X\*соответственно (в общем случае, не может быть меньше времени выполнения программы).

Следовательно, относительный выигрыш по оперативности предложенного метода верификации (по отношению к методу тестирования программ на основе ее эталонных значений):



Так как, коэффициент Kgh < 1, а c http://citforum.ru/security/articles/kazarin/m.gif 2, то получаем относительный выигрыш по оперативности испытания расчетных программ указанного типа (обладающих свойством случайной самосводимости) более чем в 1.5 раза.

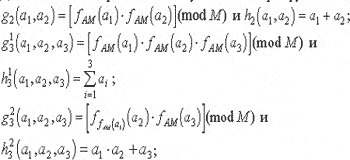
Исследования процесса верификации расчетных программ

В качестве примера работоспособности предложенного метода рассмотрим верификацию программы вычисления функции дискретного возведения в степень:

y = fAM(x) = Ax modulo M.

Выбор данной функции обусловлен тем, что она является одной из основных функций в различных теоретико-числовых конструкциях, например, в схемах электронной цифровой подписи, системах открытого распределения ключей и т.п. Это, в свою очередь, демонстрирует возможность применения предложенного метода при исследовании расчетных программ, решающих конкретные прикладные задачи. Кроме того, очевидно, что данная функция обладает свойством случайной самосводимости, а исходя из результатов работы  можно показать, что для данной функции существует (1/288, 1/8)-самотестирующаяся программа.

Для экспериментальных исследований была выбрана программа EXP из библиотеки базовых криптографических функций CRYPTOOLS, которая реализует функцию дискретного возведения в степень (размерность переменных и констант не превышает 64 байтов). Была разработана интегрированная оболочка для проведения верификации, включающая интерфейс с пользователем и программные процедуры, реализующие некоторую совокупность проверочных тестов. Экспериментальные исследования состояли из определения временных характеристик процесса верификации на основе использования ST-пары функций и определения возможности обнаружения предложенным методом преднамеренно внесенных программных ошибок. Для этого были определены следующие ST-пары функций:



В процессе исследований менялась используемая ST-пара функций и варьировалась размерность параметров A, M и аргумента X. Результаты экспериментов полностью подтвердили приведенные выше временные зависимости (технические результаты исследований авторы в данной работе опускают).

Исследование возможности обнаружения предложенным методом преднамеренно внесенных ошибок заключалось в написании программы EXPZ. Спецификация для программ EXP и EXPZ одна и та же, отличие же заключается в том, что программа EXPZ содержит программную закладку деструктивного характера. Преднамеренно внесенная закладка при исследованиях гарантировала сбой работы программы вычисления значения функции y = fAM(x) = Axmodulo M (то есть обеспечивала получение неправильного значения функции) примерно на каждой восьмой части входных значений экспоненты x.

Было проведено свыше 2000 экспериментов. Все входные значения, на которых произошел сбой программы, были обнаружены, что в дальнейшем подтвердилось проверочными тестами, основанными на использовании малой теоремы Ферма и теореме Эйлера. Этот факт, в свою очередь, экспериментально показал, что программа, реализующая алгоритм ST, является (1/8,1/288)-самотестирующейся программой.

Таким образом, предложенный метод позволяет в значительной степени сократить время испытания расчетных программ на предмет выявления непреднамеренных и преднамеренных программных дефектов. При этом по результатам испытаний можно получить количественные оценки вероятности наличия программных дефектов в верифицируемой расчетной программе. Однако, разработка формальных методов получения ST-пары функций для заданной расчетной программы, а также разработка методик ее испытания с помощью рассмотренного алгоритма требуют дальнейших теоретических и прикладных исследований.

Метод создания самокорректирующейся процедуры вычисления теоретико - числовой функции дискретного экспоненцирования

Обозначения и определения для функции дискретного возведения в степень вида gxmoduloM. Пусть In=Zqпредставляет собой множество {1,...,q}, где q= φ(M) - значение функции Эйлера для модуля M, а Z\*M - множество вычетов по модулю M, где n= |log2M|. Пусть распределение Dp есть равномерное распределение вероятностей. Тогда равновероятный случайный выбор элемента a из множества Ω будет обозначаться как ahttp://citforum.ru/security/articles/kazarin/e.gifRΩ . Оракульной программой, в данном случае, является программа вычисления функции gxmoduloM, по отношению к исследуемым самотестирующейся и самокорректирующейся программам.

Алгоритм Axmodulo N можно вычислить многими способами, один из наиболее общеизвестных и применяемых, - это алгоритм, основанный на считывании индекса (значения степени) слева направо. Этот метод достаточно прост и нагляден, история его восходит еще к 200 г. до н.э. Пусть x[1,...,n] - двоичное представление положительного числа x и A, B и N - положительные целые числа в r-ичной системе счисления, тогда псевдокод алгоритма Axmodulo N, реализованного программой Exp(') имеет следующий вид.

***Псевдокод алгоритма AxmoduloN***

Program Exp(x,N,A,R); {вход x,N,A, выход R}

begin

B:=1;

for i:=1 to n do

begin

B:=(B\*B)modN;

if [i]=1 then B:=(A\*B)modN;

end;

R:=B;

end.

Из описания алгоритма видно, что число обращений к операции вида (A\*B)moduloN будет logx+ β(x), где β(x) число единиц в двоичном представлении операнда x или вес Хэмминга x.

Построение самотестирующейся/самокорректирующейся программной пары для функции дискретного экспоненцирования.

Сначала рассмотрим следующие 4 алгоритма (см. рис.2.6 - 2.9). Для доказательства полноты и безопасности указанной пары доказывается следующая теорема.

**Теорема 2.2.** Пара программ S\_K\_exp(x,M,q,g,β) и S\_T\_exp(x,M,q,g,β) является (1/288,1/8,1/8)-самокорректирующейся/ самотестирующейся программной парой для функции gxmoduloM, с входными значениями, выбранными случайно и равновероятно из множества In.

Для доказательства теоремы необходимо доказать следующие две леммы.

**Лемма 2.1.** Программа S\_K\_exp(M,q,g,β) является (1/8)-самокорректирующейся программой для вычисления функции gxmoduloM в отношении равномерного распределения Dn.

Доказательство. Полнота программы S\_K(') означает, что если оракульная программа P(x), обозначаемая как Exp(') выполняется корректно, то и самокорректирующаяся программа S\_K(') будет выполняться кор-ректно. В данном случае полнота программы очевидна. Если P(x) корректно вычислима, то из http://citforum.ru/security/articles/kazarin/14.gif следует, что

http://citforum.ru/security/articles/kazarin/15.gif

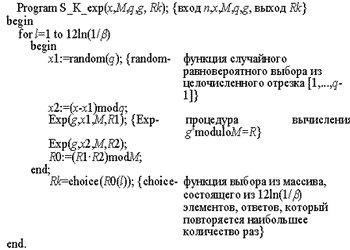


Рис. 2.6. Псевдокод алгоритма S\_K\_exp

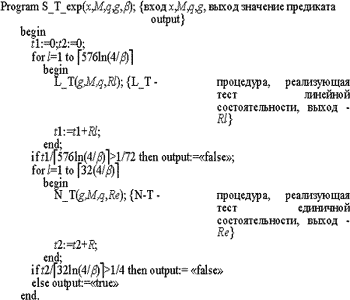


Рис. 2.7. Псевдокод алгоритма S\_T\_exp

Program L\_T(n,R); {вход g,M,q, выход Rl}

begin

x1:=random(q);

x2:=random(q);

x:=(x1+x2)modq;

Exp(g,x1,M,R1);

Exp(g,x2,M,R2);

Exp(g,x,M,R);

if R1 R2=R then Rl:=1

else Rl:=0;

end.

Рис. 2.8. Псевдокод алгоритма теста линейной состоятельности L\_T

Program N\_T(n,R); {вход g,M,q, выход Re}

begin

x1:=random(q);

x2:=(x1+1)modq;

Exp(g,x1,M,R1);

Exp(g,x2,M,R2);

if R1 g=R2 then Re:=1

else Re:=0;

end.

Рис. 2.9. Псевдокод алгоритма теста единичной состоятельности N\_T

Для доказательства безопасности сначала необходимо отметить, что так как x1http://citforum.ru/security/articles/kazarin/e.gif RZq, то и x2 имеет равномерное распределение вероятностей над Zq. Так как вероятность ошибки ε http://citforum.ru/security/articles/kazarin/b.gif 1/8, то в одном цикле вероятность Prob[Rk=fM,g(x)]http://citforum.ru/security/articles/kazarin/m.gif 3/4. Чтобы вероятность корректного ответа была не менее чем 1- β, исходя из оценки Чернова, необходимо выполнить не менее 12ln(1/β ) циклов

**Лемма 2.2.** Программа S\_T\_exp(n,M,q,g,β) является (1/288,1/8)-самотестирующейся программой, которая контролирует результат вычисления значения функции gxmoduloM с любым модулем M.

**Доказательство.** Полнота теста линейной состоятельности доказывается аналогично доказательству полноты в лемме й, где x1,x2 http://citforum.ru/security/articles/kazarin/e.gif RZq. Полнота теста единичной состоятельности вытекает исходя из следующего очевидного факта. Корректное выполнение теста [PM,g(x1)'PM.g(1)](mod M) соответствует вычислению функции:

http://citforum.ru/security/articles/kazarin/18.gif

Для доказательства условия самотестируемости необходимо отметить, что так как и в лемме 1 для того, чтобы вероятность корректных ответов Rl и Re в обоих тестах была не более 1-β достаточно выполнить тест линейной состоятельности 576ln(4/β) раз и тест единичной состоятельности 32ln(4/β) раз.

Можно показать, основываясь на теоретико - групповых рассуждениях, что возможно обобщение программы S\_T(**'**) и для других групп (алгоритмы данной программы основываются на вычислениях в мультипликативной группе вычетов над конечным полем). То есть для всех y*http://citforum.ru/security/articles/kazarin/e.gif*G, P(y)*http://citforum.ru/security/articles/kazarin/e.gif*G\*, где G\* -представляет собой любую группу, кроме групп G\*\*. К группам последнего вида относятся бесконечные группы, не имеющие конечных подгрупп за исключением {О`}, где О` - тождество группы. Таким образом, можно показать (при использовании в вышеуказанных алгоритмах параметров с их независимым, равновероятным и случайным выбором), что программа вида S\_T(**'**) является (ε/36,ε)-самотестирующейся программой. Отсюда следует доказательство утверждения леммы

Исходя из определения самотестирующейся/ самокорректирующейся программной пары и основываясь на результатах доказательств лемм 1 и 2, очевидным образом следует доказательство теоремы 1.

Замечания. Как видно из псевдокода алгоритма AxmoduloN в нем используется операция ABmodulo N. Используя ту же технику доказательств, как и для функции дискретного возведения в степень, можно построить (1/576,1/36,1/36)-самокорректирующуюся/ самотестирующуюся программную пару для вычисления функции модулярного умножения. Это справед-ливо исходя из следующих соображений. Вычисление функции fM(ab)=fM((a1+a2)(b1+b2)) следует из корректного выполнения программы с 4-х кратным вызовом оракульной программы P(a,b), то есть:

[PM(a1,b1)+PM(a1,b2)+PM(a2,b1)+PM(a2,b2)](mod M).

Алгоритм вычисления Axmodulo N выполняется для c=2. Однако, декомпозиция x, как следует из свойства самосводимости функции Axmodulo N, может осуществляться на большее число слагаемых. Хотя это приведет к гораздо большему количеству вызовов оракульной программы, но в то же время позволит значительно снизить вероятность ошибки.

Применение самотестирующихся и самокорректирующихся программ

Применение самотестирующихся, самокорректирующихся программ и их сочетаний возможно в самых различных областях вычислительной математики, а, следовательно, в самых разнообразных областях человеческой деятельности, где широко применяются вычислительные методы. К ним относятся такие направления как цифровая обработка сигналов (а, следовательно, решение проблем в системах распознавания изображений, голоса, в радио- и гидроакустике), а также методы математического моделирования процессов изменения народонаселения, экономических процессов, процессов изменения погоды и т.п. Идеи самотестирования могут найти самое широкое применение в системах защиты информации, например, в системах открытого распределения ключей, в криптосистемах с открытым ключом, в схемах идентификации пользователей вычислительных систем и аутентификации данных, где базовые вычислительные алгоритмы обладают некоторыми алгоритмическими свойствами, например, свойством случайной самосводимости, описанным выше.

Активными исследованиями в области создания самотестирующихся и самокорректирующихся программ в вычислительной математике ученые и практики начали заниматься с начала 90-х годов. В этот период были разработаны самотестирующиеся программы для ряда теоретико-числовых и теоретико-групповых задач, для решения задач с матрицами, полиномами, линейными уравнениями, рекуррентными соотношениями и аппроксимирующими функциями.

Основная идея использования идей самотестирования в криптографии заключается в неформальном девизе "Защитить самих защитников". Так как криптографические методы используются для высокоуровневого обеспечения конфиденциальности и целостности информации, собственно программно-техническая реализация этих методов должна быть свободна от программных и/или аппаратных дефектов. Таким образом, самотестирование и самокоррекция программ может эффективно применяться в современных системах защиты информации от несанкционированного доступа.

К другим направлениям в теории и практике самотестирования можно отнести построение псевдослучайных генераторов, в котором центральным звеном является конструкция, которую можно рассматривать как самокорректирующую программу для решения задачи, эквивалентной проблеме дискретных логарифмов, а также разработку самотестирующихся программ для параллельных вычислений, где используется идея самотестирования для построения схемы константной глубины для проверки мажоритарных функций.

###### 2.4.3. Создание безопасного программного обеспечения на базе методов теории конфиденциальных вычислений

Постановка задачи

Задачу конфиденциальных вычислений, которая решается посредством многостороннего интерактивного протокола можно описать в следующей постановке. Имеется n участников протокола или n процессоров вычислительной системы, соединенных сетью связи. Изначально каждому процессору известна своя "часть" некоторого входного значения x. Требуется вычислить f(x), f - некоторая известная всем участникам вычислимая функция, таким образом, чтобы выполнились следующие требования:

-корректности, когда значение f(x) должно быть вычислено правильно, даже если некоторая ограниченная часть участников произвольным образом отклоняется от предписанных протоколом действий;  
-конфиденциальности, когда в результате выполнения протокола не один из участников не получает никакой дополнительной информации о начальных значениях других участников (кроме той, которая содержится в вычисленном значении функции).

Можно представить следующий сценарий использования этой модели для разработки безопасного программного обеспечения. Имеется некоторый процесс, для управления которым необходимо вычислить функцию f. При этом последствия вычисления неправильного значения таковы, что представляется целесообразным пойти на дополнительные затраты, связанные с созданием сети из n процессоров и распределенного алгоритма для вычисления f. В системе имеется еще один абсолютно надежный участник, который имеет доступ к секретному значению x и имеет возможность выделить каждому участнику свою "долю" x. Название протоколы "конфиденциальное вычисление функции" отражает тот факт, что требование конфиденциальности является основным, то есть значение x не должно попасть в руки злоумышленника.

Общие замечания по проблематике конфиденциальных вычислений

Задача конфиденциального вычисления была впервые сформулирована А.Яо для случая с двумя участниками в 1982 г. В 1987 г. О. Голдрайх, С. Микали и А. Вигдерсон показали, как безопасно вычислить любую функцию, аргументы которой распределены среди участников в вычисли-тельной установке (то есть в конструкции, где потенциальный противник ограничен в действиях вероятностным полиномиальным временем). В их работе рассматривалась синхронная сеть связи из n участников, где каналы связи небезопасны и стороны, также как и противник, ограниченны в сво-их действиях вероятностным полиномиальным временем. В своей модели вычислений они показали, что в предположении существования односторонних функций с секретом можно построить многосторонний протокол (с n участниками) конфиденциальных вычислений любой функции в присутствие пассивных противников (т.е., противников, которым позволено толь-ко "прослушивать" коммуникации). Для некоторых типов противников (для византийских сбоев) авторы привели протокол для конфиденциального вычисления любой функции, где ( n/2 -1) участников протокола являются нечестными (или ( n/2 -1)-протокол конфиденциальных вычислений).

В дальнейшем изучались многосторонние протоколы конфиденциальных вычислений в модели с защищенными каналами. Было показано, что если противник пассивный, то существует ( n/2 -1)-протокол для конфиденциального вычисления любой функции. Если противник активный (т.е., противник, которому позволено вмешиваться в процесс вычислений), тогда любая функция может быть вычислена посредством ( n/3 -1)-протокола конфиденциальных вычислений. Эти протоколы являются безопасными в присутствие неадаптивных противников (т.е., противников, действующих в схеме вычислений, в которой множество участников независимо, но фиксировано).

В последнее время исследуются вычисления для случая активных противников, ограниченных в работе вероятностным полиномиальным временем, где часть участников вычислений может быть "подкуплена" противником и многосторонние конфиденциальные вычисления при наличии незащищенных каналов и с вычислительно неограниченным против-ником, а также исследовались многосторонние конфиденциальные вычисления с нечестным большинством участников при наличии защищенных каналов. Кроме того, изучаются многосторонние протоколы конфиденциальных вычислений при наличии защищенных каналов и динамического противника (т.е., противника, который может "подкупать" различных участников в разные моменты времени). В фундаментальной работе предложены определения для многосторонних конфиденциальных вычислений при наличии защищенных каналов и в присутствии адаптивных противников.

В работе, авторы начали комплексные исследования асинхронных конфиденциальных вычислений. Они рассмотрели полностью асинхронную сеть (т.е., сеть, где не существует глобальных системных часов), в которой стороны соединены защищенными каналами связи. Автор привел первый асинхронный протокол византийских соглашений с оптимальной устойчивостью, где противник может "подкупать" n/3 -1 из n участников вычислений.

Определение односторонней функции, описание используемых примитивов, схем и протоколов

В качестве одного из основных математических объектов в данной работе используются односторонние функции, то есть вычислимые функции, для которых не существует эффективных алгоритмов инвертирования. Необходимо отметить, что односторонняя функция - гипотетический объект, поэтому называть конкретные функции односторонними математически некорректно. Впервые такую гипотетическую конструкцию для конкретного криптографического приложения, - открытого распределения ключей, предложили У. Диффи и М. Хеллман в 1976 г.. Они показали, что вычисление степеней в мультипликативной группе вычетов над конечным полем является простой задачей с точки зрения состава необходимых вычислений, в то время как из-влечение дискретных логарифмов над этим полем - предположительно сложная вычислительная задача.

Неформально говоря, для двух независимых множеств Xи Y функция f:X- > Y называется односторонней, если для каждого xhttp://citforum.ru/security/articles/kazarin/e.gif X можно легко вычислить f(x), в то время как почти для всех y*http://citforum.ru/security/articles/kazarin/e.gif*Y вычислительно трудно по-лучить такой x*http://citforum.ru/security/articles/kazarin/e.gif*X, что f(x)=y, при условии , что такой x существует.

Далее приводится формальное определение односторонней функции в терминах теории сложности вычислений. Для этого введем следующее обозначение. Пусть a http://citforum.ru/security/articles/kazarin/e.gif RM обозначает, что элемент aслучайно, с равномерным распределением вероятностей и независимо от других событий выбран из всех элементов множества M.

Функция f называется односторонней, если существует полиномиальная машина Тьюринга T, которая на любом входе x http://citforum.ru/security/articles/kazarin/e.gif http://citforum.ru/security/articles/kazarin/z.gif вычисляет f(x) и для любого полиномиального алгоритма A справедливо следующее.

Пусть x http://citforum.ru/security/articles/kazarin/e.gif Rhttp://citforum.ru/security/articles/kazarin/z.gif n и слово y=f(x) подано на вход алгоритма A. Тогда для любого полинома p и для всех достаточно больших n вероятность Prob(f(A(y))=y)<1/p(n).

Вероятность берется по выбору значения x из http://citforum.ru/security/articles/kazarin/z.gif n и, если A - это вероятностная машина Тьюринга, - по вырабатываемым ею случайным величинам.

(n,t)-Пороговые схемы. Используемая в данном разделе (n,t)-пороговая схема, известная как схема разделения секрета Шамира, - это протокол между n+1 участниками, в котором один из участников, именуемый дилер - **Д**, распределяет частичную информацию (доли) о секрете между n участниками так, что:

* любая группа из менее чем t участников не может получить любую информацию о секрете;
* любая группа из не менее чем t участников может вычислить сек-рет за полиномиальное время.

Пусть секрет s - элемент поля **F**. Чтобы распределить s среди участников P1,...,Pn, (где n< |**F**| ) дилер выбирает полином f*http://citforum.ru/security/articles/kazarin/e.gif*F[x] степени не более t-1, удовлетворяющий f(0)=s. Участник Pi получает si=f(xi)как свою долю секрета s, где xi*http://citforum.ru/security/articles/kazarin/e.gif****F***\{0} - общедоступная информация для Pi(xi*http://citforum.ru/security/articles/kazarin/nerav.gif*xj для i*http://citforum.ru/security/articles/kazarin/nerav.gif*j).

Вследствие того, что существует один и только один полином степени не менее k-1, удовлетворяющий f(xi)=si для k значений i, схема Шамира удовлетворяет определению (n,t)-пороговых схем. Любые tучастников могут найти значение f по формуле:

http://citforum.ru/security/articles/kazarin/19.gif

Следовательно

http://citforum.ru/security/articles/kazarin/20.gif

где a1,...,ak получаются из

http://citforum.ru/security/articles/kazarin/21.gif

Таким образом, каждый ai является ненулевым и может быть легко вычислен из общедоступной информации. Проверяемая схема разделения секрета. Пусть имеется n участников вычислений и t\*(значение t\* не более порогового значения t) из них могут отклоняться от предписанных протоколом действий. Один из участников назначается дилером **Д**, которому (и только ему) становится известен секрет (секретная информация) s. На первом этапе дилер вне зависимости от действий нечестных участников осуществляет привязку к уникальному параметру u. Идентификатор дилера известен всем абонентам системы. На втором этапе осуществляется открытие (восстановление) секрета s всеми честными участниками системы. И если дилер **Д** - честный, то s=u.

Проверяемая схема разделения секрета **ПРС** представляет собой пару многосторонних протоколов (**РзПр,ВсПр**), - а именно протокола разделения секрета и проверки правильности разделения **РзПр** и протокола восстановления и проверки правильности восстановления секрета **ВсПр**, при реализации которых выполняются следующие условия безопасности. Условие полноты. Для любого s, любой константы c>0 и для достаточно больших n вероятность Prob((n,t,t\*)**РзПр**=(**Д**а,s) t\* < t & **Д** - честный)>1-n-c.

Условие верифицируемости. Для всех возможных эффективных алгоритмов **Прот**, любой константы c>0и для достаточно больших nвероятность

Prob((t\*,(Да,u))**ВсПр**=(s=u)| (n,t,t\*)**РзПр**=(Да,u)& t\* < t& **Д** - честный)<n-c.

Условие неразличимости. Для секрета s\**http://citforum.ru/security/articles/kazarin/e.gif*RS вероятность

Prob(s\*=s |(n,t,t\*)**РзПр**=(Да,s\*) & **Д** - честный)< 1/ |S| .

Свойство полноты означает, что если дилер **Д** честный и количество нечестных абонентов не больше t, тогда при любом выполнении протокола **РзПр** завершится корректно с вероятностью близкой к 1. Свойство верифицируемости означает, что все честные абоненты выдают в конце протокола **ВсПр** значение u, а если **Д** - честный, тогда все честные абоненты восстановят секрет s=u. Свойство неразличимости говорит о том, что при произвольном выполнении протокола **РзПр** со случайно выбранным секретом s\*, любой алгоритм **Прот** не может найти s\*=sлучше, чем простым угадыванием.

Широковещательный примитив (Br-субпротокол). Введем следующее определение. Протокол называетсяt-устойчивым широковещательным протоколом, если он инициализирован специальным участником (дилером **Д**), имеющим вход m(сообщение, предназначенное для распространения) и для каждого входа и коалиции нечестных участников (числом не более t) выполняются условия.

Условие завершения. Если дилер **Д** - честный, то все честные участники обязательно завершат протокол. Кроме того, если любой честный участник завершит протокол, то все честные участники обязательно завершат протокол.

Условие корректности. Если честные участники завершат протокол, то они сделают это с общим выходом m\*. Кроме того, если дилер честный, тогда m\*=m.

Необходимо подчеркнуть, что свойство завершения является более слабым, чем свойство завершения в византийские соглашения. Для Br-протокола не требуется, чтобы несбоящие процессоры завершали протокол, в том случае, если дилер нечестен.

**Br-протокол**

Код для дилера (по входу m):

1. Послать сообщение (сбщ,m) всем процессорам и завершить протокол с выходом m.  
   Код для процессоров:
2. После получения первых сообщений (сбщ,m) или (эхо,m), послать (эхо,m) ко всем процессорам и завершить протокол с выходом m.

**Предложение 2.1.** Br-протокол является t-усточивым широковещательным протоколом для противников, которые могут приостанавливать отправку сообщений.

**Доказательство.** Если дилер честен, то все несбоящие процессоры получают сообщение (сбщ,m) и, таким образом, завершают протокол с выходом m. Если несбоящий процессор завершил протокол с выходом m, то он посылает сообщение (эхо,m) ко всем другим процессорам и, таким образом, каждый несбоящий процессор завершит протокол с выходом m. Ниже описывается http://citforum.ru/security/articles/kazarin/22.gif-устойчивый широковещательный протокол, который именуется **BB**. В протоколе принимается, что идентификатор дилера содержится в параметре m.

**Протокол BB**

Код для дилера (по входу m):

1. Послать сообщение (сбщ,m) ко всем процессорам.  
   Код для процессора Pi:
2. После получения сообщения (сбщ,m`), послать сообщение (эхо,m`) ко всем процессорам.
3. После получения n-1 сообщений (сбщ,m` ), которые согласованы со значением m` послать сообщение (гот,m` ) ко всем процессорам.
4. После получения t+1 сообщений (гот,m` ), которые согласованы со значением m` , послать (гот,m` ) ко всем процессорам.
5. После получения n-1 сообщений (сбщ,m` ), которые согласованы со значением m , послать сообщение (OK,m`) ко всем процессорам и принять m` как распространяемое сообщение.

Протокол византийского соглашения (BA-субпротокол). Введем следующее определение. При византийских соглашениях для любого начального входа xi, ihttp://citforum.ru/security/articles/kazarin/e.gif [1,...,n] участника i и некоторого параметраd (соглашения) должны быть выполнены следующие условия.

Условие завершения. Все честные участники вычислений в конце протокола принимают значение d.

Условие корректности. Если существует значение x такое, что для честных участников xi=x, тогда d=x.

Обобщенные модели вычислений для сети синхронно и асинхронно взаимодействующих процессоров

Принимаемая модель создания программного обеспечения на базе методов конфиденциального вычисления функции позволяет единообразно трактовать как ошибки, возникающие, например, в результате сбоев технических средств, так и ошибки, возникающие за счет привнесения в вычислительный процесс преднамеренных программных дефектов. Следует отметить, что протоколы конфиденциального вычисления функции относятся к протоколам, которые предназначены, прежде всего, для защиты процесса вычислений от действия "разумного" злоумышленника, то есть от злоумышленника, который всегда выбирает наихудшую для нас стратегию.

В модели вводится дополнительный параметр t, t < n, - максимальное число участников, которые могут отклоняться от предписанных протоколом действий, то есть максимальное число злоумышленников. Поскольку злоумышленники могут действовать заодно, обычно предполагается, что против протокола действует один злоумышленник, который может захватить и контролировать любые t из n процессоров по своему выбору.

Модель сбоев и модель противника

Рассматривается сеть взаимодействующих процессоров. Некоторые процессоры могут "сбоить". При сбоях, приводящих к останову (FS-сбоях), сбоящий процессор может приостановить в некоторый момент времени отправку своих сообщений. В то же время предполагается, что сбоящие процессоры продолжают получение сообщений и могут выдавать "некую информацию" в свои выходные каналы. При Византийских сбоях (By-сбоях) процессоры могут произвольным образом сотрудничать друг с другом с целью получения необходимой для них информации или с целью нарушения процесса вычислений. При By-сбоях сбоящие процессоры могут объединять свои входы и изменять их. В то же время это должно происходить при условии невозможности изучения любой информации о входах не сбоящих процессоров.

Сбои могут быть статическими и динамическими. При статических сбоях множество сбоящих процессоров фиксировано в начале и процессе вычислений, при динамических - множество сбоящих процессоров может меняться, как может меняться и характер самих сбоев. Сбоящие процессоры будут также называться нечестными участниками вычислений, в противоположность честным участникам, которые выполняют предписанные вычисления.

Противника можно представлять как некий универсальный процессор, действия которого заключаются в стремлении "сделать" процессоры сети сбоящими ("подкупить" их).

По аналогии со сбоями противник может быть динамическим и статическим. Статическим противником является противник, который "сотрудничает" с фиксированным количеством сбоящих процессоров ("подкупленных" противником). При действиях динамического противника количество сбоящих процессоров может меняться (в том числе непредсказуемым образом).

Кроме того, противник может быть активным и пассивным, а также с априорными и апостериорными протокольными и раундовыми действиями. Пассивный противник не может изменять сообщения, циркулируемые в сети. Активный противник может знать все о внутренней конфигурации сети, может читать и изменять все сообщения сбоящих процессоров и может выбирать сообщения, которые будут посылать сбоящие процессоры при вычислениях. Активный динамический противник может в начале каждого раунда "подкупить" несколько новых процессоров. Таким образом, он может изучить информацию, получаемую ими в текущем раунде, и принять решение "подкупить" ли ему новый процессор, или нет. Такой противник может собирать и изменять все сообщения от сбоящих процессоров к несбоящим. В то же время гарантируется, что все сообщения (в том числе, между честными процессорами) будут доставлены адресатам в конце текущего раунда. Действия аналогичного характера активный противник может выполнить не только в течение раунда (в его начале и конце), но и до начала выполнения протокола и после его завершения.

Противник называется t-противником, если он сотрудничает с t процессорами.

Модель взаимодействия

Взаимодействие процессоров может осуществляться посредством конфиденциальных и открытых каналов связи. Каждые два процессора могут быть иметь симплексное, полудуплексное или дуплексное соединение. При этом каждое из соединений, в зависимости от решаемой задачи, в некоторый промежуток времени, может быть либо конфиденциальным, либо открытым. Кроме того, может существовать широковещательный канал связи, то есть канал, посредством которого один из процессоров может одновременно распространить свое сообщение всем другим процессорам вычисли-тельной системы. Такой канал еще называется каналом с общей шиной.

Взаимодействие в сети характеризуется трафиком T, который может представлять собой совокупность сообщений, полученных участниками вычислений в r-том раунде в установленной модели взаимодействия.

Синхронная модель вычислений

Рассматривается сеть процессоров, функционирование которой синхронизируется общими часами (синхронизатором). Каждое локальное (внутреннее) вычисление соответствует одному моменту времени (одному такту часов). В данной модели отрезок времени между i и i+1 тактами называется раундом при синхронных вычислениях. За один раунд протокола каждый процессор может получать сообщения от любого другого процессора, выполнять локальные (внутренние) вычисления и посылать сообщения всем другим процессорам сети. Имеется входная лента "только-для-чтения", которая первоначально содержит строку Λ (k) (например вида 1k), при этом k является параметром безопасности системы. Каждый процессор имеет ленту случайных значений, конфиденциальную рабочую ленту "только-для-чтения" (первоначально содержащую строку Λ), конфиденциальную входную ленту "только-для-чтения" (первоначально содержащую строку xi), конфиденциальную выходную ленту "только-для-записи" (первоначально содержащую строку Λ) и несколько коммуникационных лент. Между каждой парой процессоров i и j существует конфиденциальный канал связи, посредством которого i посылает безопасным способом сообщение процессору j. Данный канал (коммуникационная лента) исключает запись для iи исключает чтение для j. Каждый процессор i имеет также широковещательный канал, представляющий собой ленту, исключающую запись для i и являющийся каналом "только-для-чтения" для всех остальных процессоров сети.

Предполагается, что в раунде r для каждого процессора i существует однозначно определенное сообщение (возможно пустое), которое распространяет процессорi в этом раунде и для каждой пары процессоров i и j существует единственное сообщение, котороеi может безопасным образом послать j в данном раунде. Все каналы являются помеченными так, что каждый получатель сообщения может идентифицировать его отправителя.

Процессор i запускает программу πi, совокупность которых, реализует распределенный алгоритм **П**. Протоколом работы сети называется n-элементный кортеж ***P***=( π1, π2,.., πn). Протокол называется t-устойчивым, если t процессоров являются сбоящими во время выполнения протокола. Историей процессора i являются: содержание его конфиденциального и общего входа, все распространяемые им сообщения, все сообщения, полученные i по конфиденциальным каналам связи, и все случайные параметрами, сгенерированные процессором i во время работы сети.

Идеальный и реальный сценарии

Для доказуемо конфиденциального вычисления вводятся понятия идеального и реального сценария . В идеальном сценарии дополнительно вводится достоверный процессор (TP-процессор). Процессоры конфиденциально посылают свои входы ТР-процессору, который вычисляет необходимый результат (выход) и также конфиденциально посылает его обратно процессорам сети. Противник может манипулировать с этим результатом (вычислить или изменить его) следующим образом. До начала вычислений он может подкупить один из процессоров и изучить его секретный вход. Основываясь на этой информации, противник может подкупить второй процессор и изучить его секретный вход. Это продолжается до тех пор пока противник не получит всю необходимую для него информацию. Далее у противника есть два основных пути. Он может изменить входы нечестных процессоров. После чего те, вместе с корректными входами честных процессоров, направляют свои новые измененные входы ТР-процессору. По получению от последнего выходов (значения вычисленной функции) противник может приступить к изучению выхода каждого нечестного процессора. Второй путь заключается в последовательном изучении входов и выходов процессоров, подключая их всякий раз к числу нечестных. В данном случае рассматривается противник, который не только может изучать входы нечестных процессоров, но и менять их, изучать по полученному значению функции входы честных процессоров.

В реальном сценарии не существует ТР-процессора, и все участники вычислений моделируют его поведение посредством выполнения многостороннего интерактивного протокола.

Грубо говоря, считается, что вычисления в действительности (в реальном сценарии) безопасны, если эти вычисления "эквивалентны" вычислениям в идеальном сценарии. Точное определение (формальное определение) понятия эквивалентности в этом контексте является одной из основных проблем в теории конфиденциальных вычислений.

Асинхронная модель вычислений

В данном подразделе рассматривается полностью асинхронная сеть из n процессоров, которые соединены открытыми или конфиденциальными каналами. При этом не существует единых глобальных часов. Любое сообщение в сети может быть задержано независимым образом. В то же время считается, что каждое посланное сообщение обязательно будет получено адресатом. Вопрос о переупорядочивании сообщений не исследуется.

Вычисления в асинхронной модели рассматривается как последовательность шагов. На каждом шаге активизируется один из процессоров. При этом активизация процессора происходит по получении им сообщения. После чего он выполняет внутренние вычисления и возможно выдает сообщения в свои выходные каналы. Порядок шагов определяется Планировщиком (D), неограниченным в вычислительной мощности. Каналы, являющиеся конфиденциальными или открытыми, рассматриваются как частные планировщики (PDj). Информация, известная частному планировщику, есть не что иное как сообщения, посланные от источника сообщений их адресату. В данной модели вычислений каждый их шах рассматривается как раунд при асинхронных вычислениях. Идеальный и реальный сценарии в асинхронной модели

Сценарий в асинхронной модели с ТР-участником заключается в добавлении ТР-процессора в существующую асинхронную сеть при наличии t потенциальных сбоев (сбоящих процессоров). При этом честные процессоры, также как и ТР-процессор не могут ожидать наличия более, чем n-t входов для вычислений с целью получения их выхода, так как t процессоров (сбоящие процессоры) могут никогда не присоединиться к вычислениям.

Асинхронный ТР-сценарий

В начале вычислений процессоры посылают свои входы ТР-процессору. В то же время, существует планировщик D, который доставляет сообщения участников некоторому базовому подмножеству процессоров, мощностью не меньше n-t, обозначаемому как C и являющемуся независимым от входов честных процессоров. ТР-процессор по получению входов - аргументов функции (возможно некорректных) из множества C, предопределенно оценивает значение вычисляемой функции, основываясь на C и входах участников из С. (Здесь для корректности может использоваться следующее предопределенное оценивание: установить входы из C в 0 и вычислить данную функцию). Затем ТР-процессор посылает значение оценочной функции обратно участникам совместно с базовым множеством С. Наконец честные процессоры вычислений выдают то, что они получили от ТР-участника. Нечестные участники выдают значение некоторой независимой функции, информацию о которой они "собирали" в процессе вычислений. Эта информация может состоять из их собственных входов, случайных значений, используемых при вычислениях и значения оценочной функции.

Так же как и в синхронной модели, вычисления в реальной асинхронной модели безопасны, если эти вычисления "эквивалентны" вычислениям в ТР-сценарии.

Далее в соответствии с работой  сделаем попытку формализовать понятия полноты и безопасности протокола асинхронных конфиденциальных вычислений.

Безопасность асинхронных вычислений

Для удобства мы унифицируем коалицию нечестных процессоров и планировщика. Это не увеличит мощность противника в ТР-сценарии, так как и нечестные участники, и планировщик имеют неограниченную вычислительную мощность.

Для вектора *http://citforum.ru/security/articles/kazarin/xx.gif*=x1,...,x2 и множества http://citforum.ru/security/articles/kazarin/23.gif, пусть http://citforum.ru/security/articles/kazarin/xx.gifC определяет вектор http://citforum.ru/security/articles/kazarin/xx.gif, спроектированный на индексы из C. Для подмножества B http://citforum.ru/security/articles/kazarin/c.gif [n] и вектора |B| -vector http://citforum.ru/security/articles/kazarin/24.gif, пусть http://citforum.ru/security/articles/kazarin/25.gif определяет вектор, полученный из http://citforum.ru/security/articles/kazarin/xx.gif подстановкой входов из B соответствующими входами из http://citforum.ru/security/articles/kazarin/zz.gif. Используя эти определения, можно определить оценочную функцию f с базовым множеством C http://citforum.ru/security/articles/kazarin/c.gif [n] какhttp://citforum.ru/security/articles/kazarin/26.gif

Пусть A - область возможных входов процессоров и пусть R - область случайных входов. ТР-противник это кортеж ***А***=(B,h,c,O), где B http://citforum.ru/security/articles/kazarin/c.gif [n] - множество нечестных участников, h:A|B| x R - > A|B|-функция подстановки входов, с:A|B| x R -> {C http://citforum.ru/security/articles/kazarin/c.gif [n] | |C| http://citforum.ru/security/articles/kazarin/m.gif n-t} - функция выбора базового множества и O:A|B| x Rx A - > {0,1}\* - функция выхода для нечестных процессоров.

Функции h и O представляют собой программы нечестных процессоров, а функция c- комбинацию планировщика и программ нечестных участников вычислений.

Пусть f:An - > A для некоторой области A. Выход функции вычисления f в ТР-сценарии по входу http://citforum.ru/security/articles/kazarin/xx.gif и с ТР-противником ***А***=(B,h,c,O) - это n-vector **τ**(http://citforum.ru/security/articles/kazarin/xx.gif,**A**) = τ1(http://citforum.ru/security/articles/kazarin/xx.gif,**A**)...τn(http://citforum.ru/security/articles/kazarin/xx.gif,**A**) случайных переменных, удовлетворяющих для каждого 1 http://citforum.ru/security/articles/kazarin/b.gif i http://citforum.ru/security/articles/kazarin/b.gif n:

http://citforum.ru/security/articles/kazarin/27.gif

где r объединенный случайный вход нечестных процессоров, C= (*http://citforum.ru/security/articles/kazarin/xx.gif*B,r) и http://citforum.ru/security/articles/kazarin/28.gif

Акцентируем внимание на то, что выход сбоящих процессоров и вы-ход несбоящих процессоров вычисляется на одном и том же значении случайного входа r.

Далее формализуем понятие вычисления "в реальной жизни".

1. Пусть ***B***=(B,β) - коалиция нечестных процессоров, где B http://citforum.ru/security/articles/kazarin/cc.gif [n] - множество нечестных процессоров и β - их совместная стратегия.
2. Пусть πi(http://citforum.ru/security/articles/kazarin/xx.gif,B,D) определяет выход процессора Pi после выполнения протокола πi по входу http://citforum.ru/security/articles/kazarin/xx.gif, с планировщиком D и коалиции **B**. Пусть также π (http://citforum.ru/security/articles/kazarin/xx.gif,B,D)= π1(http://citforum.ru/security/articles/kazarin/xx.gif,B,D)... πn(http://citforum.ru/security/articles/kazarin/xx.gif,B,D).

Кроме того, пусть f:An- > A для некоторой области A и пусть π - протокол для n процессоров. Будем говорить, что π безопасно t-вычисляет функцию f в асинхронной модели для каждого частного планировщика PD и каждой коалиции ***B*** с не более, чем из t нечестных процессоров, если выполняются следующие условия.

Условие завершения (условие полноты). По всем входам все честные процессоры завершают протокол с вероятностью 1.

Условие безопасности. Существует ТР-противник ***A*** такой, что для каждого входа http://citforum.ru/security/articles/kazarin/xx.gif векторы τ(*http://citforum.ru/security/articles/kazarin/xx.gif*,A) и π(*http://citforum.ru/security/articles/kazarin/xx.gif*,B,D) идентично распределены (эквивалентны).

Конфиденциальное вычисление функции

Общие положения.Для некоторых задач, решаемых в рамках методологии конфиденциальных вычислений, достаточно введения определения конфиденциального вычисления функции.

Пусть в сети N, состоящей из n процессоров P1,P2,...,Pn со своими секретными входами x1,x2,...,xn, необходимо корректно (т.е. даже при наличии t сбоящих процессоров) вычислить значение функции(y1,y2,...,yn)=f(x1,x2,...,xn) без разглашения информации о секретных аргументах функции, кроме той, информации, которая содержится в вычисленном значении функции.

По аналогии с идеальным и реальным сценариями, приведенными выше, можно ввести понятия "реальное и идеальное вычисление функции f" .

Пусть множество входов и выходов обозначается как X и Y соответственно, размерности этих множеств |X| =X и |Y| = μ . Множество случайных параметров, используемых всеми процессорами сети, обозначается через R, размерность - |R| = v. Кроме того, через W обозначим рабочее пространство параметров сети. Через T(r) обозначается трафик в r-раунде, через ti(r) - трафик для процессора i в r-том раунде, r0 и rk - инициализирующий и последний раунды протокола соответственно и r\* - заданный неким произ-вольным образом раунд выполнения протокола ***P***.

Пусть функцию f можно представить как композицию d функций (функций от двух аргументов-векторов) http://citforum.ru/security/articles/kazarin/29.gif

http://citforum.ru/security/articles/kazarin/30.gif

Аргументы функции gη являются рабочими параметрами w1,...,wn участников протокола с трафиком(t1,...,tn) в r раунде. Значения данной функции gη являются аргументами (рабочими параметрами протокола с трафиком (t1,...,tn) в r+1 раунде) для функции gη+1.

Из определения следует, что функция f:(Xn*http://citforum.ru/security/articles/kazarin/ox.gif*Rn*http://citforum.ru/security/articles/kazarin/ox.gif*W)- > Y, где http://citforum.ru/security/articles/kazarin/ox.gif - декартово произведение множеств, реализует:

http://citforum.ru/security/articles/kazarin/31.gif

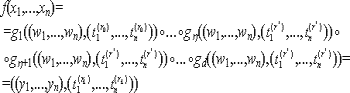
Введем понятие моделирующего устройства ***M***. Здесь можно проследить некоторые аналогии с моделирующей машиной в интерактивных сис-темах доказательств с нулевым разглашением.

Пусть **рΘПрот** - распределение вероятностей над множеством историй (трафика Т и случайных параметров) нечестных участников во время выполнения протокола Р.

Моделирующее устройство, взаимодействующее с нечестными участниками, осуществляет свое функционирование в рамках идеального сценария. Моделирующее устройство ***М*** создает распределение вероятностей параметров взаимодействия **иΘПрот** между ***М*** и нечестными участниками.

Протокол P конфиденциально вычисляет функцию f(x), если выполняются следующие условия:

align="justify"Условие корректности. Для всех несбоящих процессоров Pi функция



вычисляется с вероятностью ошибки близкой к 0.

Условие конфиденциальности. Для заданной тройки (x,r,w)http://citforum.ru/security/articles/kazarin/e.gif(Xn*http://citforum.ru/security/articles/kazarin/ox.gif*Rn*http://citforum.ru/security/articles/kazarin/ox.gif*W) распределения **рΘПрот** и **иΘПрот** являются статистически не различимыми.

Функция f, удовлетворяющая условиям предыдущего определения называется конфиденциально вычислимой.

Проверяемые схемы разделения секрета как конфиденциальное вычисление функции

Для вышеприведенной проверяемой схемы **ПРС** и обобщенных моделей противника, сбоев, взаимодействия и вычислений синтезируем схему разделения секрета, которая являлась бы работоспособной в протоколах конфиденциальных вычислений. Рассматривается полностью синхронная сеть взаимодействующих процессоров в условиях проявления By-сбоев. Противник представляется как активный динамический t-противник. Взаимодействие процессоров может осуществляться посредством конфиденциальных каналов связи. Кроме того, существует широковещательный канал связи.

Схема проверяемого разделения секрета, рассматриваемая как схема конфиденциального вычисления функции, значение которой является распределенный среди процессоров проверенный на корректность и затем восстанавливаемый секрет, обозначается как **ПРСК**.

Пусть сеть N состоит из n процессоров P1,P2,...,Pn-1,Pn, где Pn - дилер **Д** сети N. Множество секретов обозначается через S, размерность этого множества |S| =l. Множество случайных параметров, используемых всеми процессорами сети, обозначается через R, размерность |R| = v. Через Wобозначается рабочее пространство параметров сети.

Требуется вычислить посредством выполнения протокола ***P***=(**РзПр,ВсПр**) функцию f(x), где f - представляется в виде композиции двух функций g o h. Пусть функция g:(Sn*http://citforum.ru/security/articles/kazarin/ox.gif*Rn*http://citforum.ru/security/articles/kazarin/ox.gif*W) - > W, а функция h:W - > S.

Проверяемая схема разделения секрета ПРСК называется t-устойчивой, если протокол разделения секрета и проверки правильности разделения **РзПр** и протокол восстановления секрета **ВсПр** являются t-устойчивыми. Функция f является конфиденциально вычислимой, если конфиденциально вычислимы функции g иh.

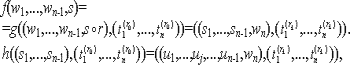
t-Устойчивая верифицируемая схема разделения секрета **ПРСК** - есть пара протоколов **РзПр** и **ВсПр**, при реализации которых выполняются следующие условия.

Условие полноты. Пусть событие A1 заключается в выполнении тождества

http://citforum.ru/security/articles/kazarin/33.gif

Тогда для любой константы c>0 и для достаточно больших n вероятность Prob(A1)>1-n-c.

Условие корректности. Пусть событие A2 заключается в выполнении тождества



где uj=s для j*http://citforum.ru/security/articles/kazarin/e.gif*G и G - разрешенная коалиция участников.

Тогда для любой константы c>0, достаточно больших n, для границы t\*< tи любого алгоритма **Прот** вероятность Prob(A2)<n-c.</n

Условие конфиденциальности.

Функция http://citforum.ru/security/articles/kazarin/35.gif -

конфиденциально вычислима.

Функция http://citforum.ru/security/articles/kazarin/36.gif - конфиденциально вычислима.

Свойство полноты означает, что если все участники протоколов **РзПр** и **ВсПр** следуют предопределенным вычислениям, тогда любая коалиция несбоящих процессоров может восстановить секрет s. Свойство корректности означает, что при действиях t-противника **Прот**, любая разрешенная коалиция из n процессоров сети может корректно разделить, проверить и восстановить секрет. Свойство конфиденциальности вытекает из свойства конфиденциальности функции.

**Схема ПРСК**

Предлагаемая схема ПРСК, является (n/3-1)-устойчивой и использует схему разделения секрета Шамира. Пусть n=3t+4 и все вычисления выполняются по модулю большого простого числа p>n. Из теории кодов с исправлением ошибок известно, что если вычисляем полином f степени t+1 в n-1различных точках i, гдеi=1,...,n-1, тогда по данной последовательности si=f(i), можно восстановить коэффициенты полинома за время, ограниченное некоторым полиномом, даже если t элементов в последовательности произвольно изменены. Это вариант кода Рида-Соломона, известный как код Берлекампа-Велча. Пусть далее K - параметр безопасности и K/n означает [K/n] .

**Протокол РзПр**

1. Дилер Д выбирает случайный полином f0(x) степени t+1 с единственным условием: f0(0)=s - его секрет. Затем он посылает процессору Pi его долю si=f0(i). Кроме того, он выбирает 2К случайных полиномов f1,...,f2K степени t+1 и посылает процессору Pi значения fj(i) для каждого j=1,...,2К.
2. Каждый процессор Pi распространяет (посредством широко-вещательного канала) K/n случайных битов α(i-1)K/n+j, для j=1,...,K/n.
3. Дилер **Д** распространяет полиномы gj=fj+ αjf0 для всехj=1,...,K.
4. Процессор Pi проверяет, удовлетворяют ли его значения полиномам, распространяемым дилером. Если он обнаруживает ошибку, он ее декларирует для всех. Если более чем t процессоров сообщают об этом, тогда дилер считается нечестным и все процессоры принимают по умолчанию значение нуля как секрет дилера **Д**.
5. Если менее чем t процессоров сообщили об ошибке, дилер распространяет значения, который он посылал в первом раунде тем процессорам, которые сообщали об ошибке дилера.
6. Каждый процессор Pi распространяет K/n случайных битов β(i-1)K/n+j для j=1,...,K/n.
7. Дилер **Д** распространяет полиномы hj=fK+j+ βjf0 для всех j=1,...,K.
8. Процессор Pi проверяет, удовлетворяют ли значения, которые он имеет, полиномам, распространяемым дилером **Д** в 5-м раунде. Если он находит ошибку, он декларирует об этом всем процессорам сети. Если более чем tпроцессоров сообщают об ошибке, тогда дилер нечестный и все процессоры принимают по умолчанию значение нуля как секрет дилера **Д**.

**Протокол ВсПр**

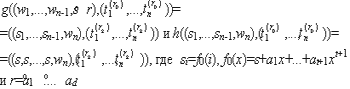
1. Каждый процессор Pi выбирает случайный многочлен hi степени t+1 такой, что hi(0)=si - его собственная входная доля секрета. Он посылает процессору Pj значение hi(j).
2. Каждый процессор Pi выбирает случайные полиномы pi(x), qi,1(x),...,qi,2K(x) степени t+1 со свободным членом 0 и посылает участнику Pj значения pi(j), qi,1(j),...,qi,2K(j).
3. Каждый процессор Pi распространяет Kслучайных битов  
   γl,(i-1)K/n+m для l=1,...,n и m=K/n.
4. Каждый процессор распространяет следующие полиномы rj=qi,j+ γi,jpi для каждого j=1,...,K.
5. Каждый процессор Pi проверяет, что информация процессора Pl, посланная ему в 1-м раунде, соответствует тому, что Pl распространяет в 3-м раунде. Если имеется ошибка или Pl распространяет полином с ненулевым свободным членом, процессор Pi распространяет сообщение badl. Если более чем t процессоров распространяют badl, процессор Pl исключается из сети и все другие процессоры принимают как 0 долю процессора Pl. В противном случае, Pl распространяет информацию, которую он посылал в раунде 1 процессорам, распространявшим сообщение badl.
6. Каждый процессор Pi распространяет K случайных битов  
   δl,(i-1)K/n+m для l=1,...,n и m=1,...,K/n.
7. Каждый процессор Pi распространяет следующие полиномы rj=qi,K+j+ δi,jpi для каждый j=1,...,K.
8. Каждый процессор Pi проверяет, что информация, посланная процессором Pl, в 1-м раунде и распространенная в 5-м раунде, соответствует полиномам процессора Pl, распространенным в 7-м раунде. Если имеется ошибка или Pl распространил полином с ненулевым свободным членом процессор Pi распространяет badlr. Если более чем t процессоров распространили badl, тогда Pl - нечестен и все процессоры принимают его долю, равную 0.
9. Каждый процессор Pl распределяет всем другим процессорам следующее значениеsi+p1(i)+p2(i)+...+p(i), затем интерполирует полином F(x)=f0(x)+p1(x)+p2(x)+...+pn(x)c использованием алгоритма с исправлением ошибок Берлекампа-Велча. Секрет будет равенs=F(0)=f(0).

Заметим, что если дилер **Д** честен, в конце протокола **ВсПр** противник, даже зная секрет s и t долей "подкупленных" процессоров, ничего не узнает о долях секрета честных процессоров, так как полином имеет степень t+1 и ему необходимо для интерполяции t +2 точки.

Доказательство безопасности схемы проверяемого разделения секрета

Теорема 2.3. Схема **ПРСК** является (n/3-1)-устойчивой.

Доказательство. Пусть



(то есть полином, созданный, с использованием случайного параметра r дилера **Д**).

При рассмотрении протокола **РзПр** напомним, что ti - трафик процессора Pi. Ясно, что для всех процессоров Pi, i*http://citforum.ru/security/articles/kazarin/b.gif*n, функция входа всегда возвращает пустую строку Ii(ti)=ε, так как процессоры не вносят никакие входы в процессе вычисления функции g. Для дилера **Д**, **Д**=Pn+1, функция входа немного сложнее. Пусть mi - сообщение, который дилер распространяет процессору Pi в 5-м раунде, если Pi сообщил об ошибке в 4-м раунде или сообщение, которое дилер послал процессору Pi в 1-м раунде, еслиPi не заявлял об ошибке. Тогда ID(tD)=f(0), где f=BW(m1,...,mn) - полином степени t+1, следующий из интерполяции Берлекампа-Велча. Функция выхода более простая: Oi(ti)=mi, где mD=ε.

При рассмотрении протокола **ВсПр**, определим вход и выход функции g. Функция входа Ii для процессораPi определена как следующая: пусть mi,j - сообщение, посланное процессором Pi процессору Pj в 1-м раунде; Ii(ti)=hi(0), где hi=BW(mi,1,...,mi,n) - многочлен степени t+1, следующий из интерполяции Берлекампа-Велча. Если такого полинома не существует, то Ii(ti)=0. Функция выхода следующая: пустьMi - сообщение, распространяемая процессором Pi в раунде 9; Oi(ti)=F(0)=s, где F=BW(M1,...,Mn) - полином степени t+1, следующий из интерполяции Берлекампа-Велча.

Далее для доказательства теоремы необходимо доказать выполнения условий полноты, корректности и конфиденциальности. Полнота. Если дилер **Д** - честный, исходя из свойств схемы **ПРСК**, любой несбоящий процессор может восстановить секрет s с вероятностью 1, так как посредством определенных выше функций gи h

http://citforum.ru/security/articles/kazarin/38.gif

реализуется

http://citforum.ru/security/articles/kazarin/39.gif

Корректность. Для доказательства теоремы необходимо доказать следующие две леммы.

Лемма 2.3. Пусть функция g имеет вид

http://citforum.ru/security/articles/kazarin/40.gif

Тогда g корректно вычисляет доли секрета s1,...,sn-1.

Доказательство. Сначала мы должны доказать, что для всех несбоящих процессоров Pi, значение Ii(ti)равно корректному входу. Если дилер **Д** - честный, то mi=f(i), где f - многочлен степени t+1 со свободным членом s (секретом). Таким образом, ID(tD)=s - если дилер честный. Второе условие корректности состоит в том, что с высокой вероятностью должно выполняться O(t)=g(I(t)). В нашем случае это означает, что с высокой вероятностью значенияmi, находящиеся у несбоящих процессоров, должны предназначаться для единственного полинома степени t+1. Это справедливо с вероятностью http://citforum.ru/security/articles/kazarin/41.gif , где не менее http://citforum.ru/security/articles/kazarin/42.gif битов выбраны действительно случайно несбоящими процессорами в раундах 2 и 6. Каждый бит представляет запрос, на который нечестный дилер, распределивший "плохие" доли, должен будет ответить правильно в следующем раунде только с вероятностью 1/2 (то есть, если он предскажет правильно значение бита). Следовательно, это и есть граница для вероятности ошибки

Лемма 2.4. Пусть функция h имеет вид

http://citforum.ru/security/articles/kazarin/43.gif

Тогда h корректно восстанавливает секрет s.

Доказательство. Понятно, что для всех несбоящих процессоров Ii(ti)=si - корректная доля входа. В этом случае необходимо проверить, что с высокой вероятностью O(t)=h(I,t), а это означает, что необходимо доказать, что

http://citforum.ru/security/articles/kazarin/44.gif

Это равенство не выполняется, если:

* любой сбоящий процессор Pl "преуспевает" в получении случайного "мусора" вместо значений pl(j)в раунде 2 (в этом случае, сообщения Mi не будут интерполировать полином);
* процессор Piраспределяет pl(j) в раунде 2 и использует полином со свободным членом, отличным от нуля (в этом случае, Mi восстановит другой секрет).

Так как мы уже знаем, что Pl "преуспевает" в любом из двух описанных случаев с вероятностью http://citforum.ru/security/articles/kazarin/45.gif, то, следовательно, имеется не более, чем n/3 сбоящих процессоров и вероятность того, что протокол вычисляет неправильный выход не более, чем n/3(http://citforum.ru/security/articles/kazarin/45.gif), что для достаточного большого K, является экспоненциально малым.

Конфиденциальность. Для доказательства теоремы необходимо доказать следующие две леммы.

Лемма 2.5. Пусть функция g имеет вид

http://citforum.ru/security/articles/kazarin/46.gif

Тогда g конфиденциально вычислима в отношение долей секрета s1,...,sn-1.

Доказательство. Доказательство условия конфиденциальности для функции g заключается в описании работы моделирующего устройства **М**, которое взаимодействует со сбоящими процессорами (в том числе с нечестным дилером) и создает "почти" такое же распределение вероятностей, которое возникает у сбоящих процессоров во время реального выполнения протокола **РзПр**.

Необходимо рассмотреть два случая.

Случай A:Дилер нечестен до начала 1-ого раунда. Моделирующее устройство будет следовать только командам процессоров с единственным исключением, что оно будет "передавать" их в какое-либо время противнику в случае "сговора". Так как процессоры не сотрудничают по любому входу, то это сводит моделирование к работе схемы проверяемого разделения секрета с нечестным дилером. Так что моделирование будет неразличимо с точки зрения противника.

Случай B: Дилер честен до начала 1-ого раунда. Моделирующее устройство в 1-м раунде будет создавать случайный "ложный" секрет s' и распределять его процессорам в соответствии с командами протокола с полиномом f'. Если дилер честен в течение всего протокола, тогда он будет выполняться с точки зрения противника как обычный протокол проверяемого разделения секрета с честным дилером. Если дилер нечестен после 1-ого раунда противник и моделирующее устройство получит из оракула истинный вход s дилера. В этом случае моделирующее устройство передает управление от дилера к противнику и меняет полином, используемый для разделения на новый многочлен f" такой, что f"(0)=s и f"(i)=f'(i) для всех процессоров Pi, которые были "подкуплены" противником. Изменения моделирующего устройства в соответствии со случайным полиномом fi, используемым для доказательств с нулевым разглашением (см, например, [[6](http://citforum.ru/security/articles/kazarin/4.shtml#6),[27](http://citforum.ru/security/articles/kazarin/4.shtml#27),[29](http://citforum.ru/security/articles/kazarin/4.shtml#29)]) делает их совместимыми с любым широковещательным каналом. Моделирующее устройство может всегда сделать это, так как противник имеет не более t точек полинома степени t+1. Далее моделирующее устройство использует полином f" для работы несбоящих процессоров, все еще находящихся под его управлением. Можно утверждать, что для противника эти вычисления не отличимы от реальных вычислений. Единственный момент, отличающийся от реальных вычислений, - это тот факт, что доли секрета, которые противник получает до того, как дилер становится нечестным, созданы с использованием другого полинома. Но благодаря свойствам полиномов - это не является проблемой для моделирующего устройства, в том случае, если дилер нечестен.

Лемма 2.6. Пусть функция h имеет вид

http://citforum.ru/security/articles/kazarin/47.gif

Тогда h конфиденциально вычислима в отношение секрета s.

Доказательство. Работа моделирующего устройства **M** сводится к следующему.

**Описание работы моделирующего устройства M**

**1.** В раунде 1 **M** моделирует процессор Pi, выбирая случайный полином h'i степени t+1 и посылает h'i(j) к Pj. В этом месте моделирующему устройству позволено получить из оракула выход функции, так что **M**будет изучать истинный секрет s. Если такой процессор является "подкупленным" противником Прот в конце этого раунда (или в следующих раундах), то и **M**, и **Прот** узнают истинную долю sl и **M** должен изменить полином h'l в соответствии с тем, что h'l(0)=sl, но без изменения значения в точках, уже известных противнику. Моделирующее устройство M может всегда cделать это, потому что противник имеет не более t точек полинома степени t+1.

**2-8**. В течение раундов 2-8 моделирующее устройство полностью следует явно командам процессоров. Так как все что делают процессоры в этих раундах полностью случайно и нет влияния на их входы, **M**будет всегда способен создать неразличимые распределения.

**9.** Моделирующее устройство **M** выбирает полином g степени t+1 такой, что g(0)=s и затем для каждого процессора Pi, устройство **M** распространяет g(i)+pi(i)+...+pn(i), где pj - полином, распределенный процессором Pj в течение раундов 2-8 моделирования. Интерполяция Рида-Соломона этих значений даст как результат секрет s. Если процессор Pl является сбоящим в конце этого раунда, тогда и **M**, и **Прот**узнают из оракула истинную долю входа sl. Если sl*http://citforum.ru/security/articles/kazarin/nerav.gif*g(l), тогда **M** только изменит значение pl в точке l так, чтобы сделать полную сумму соответствующей такой широковещательной передаче.

Моделирование неотличимо от реального выполнения с точки зрения противника. Фактически, в раундах 2-8 все сообщения случайны и не связаны с входом, так что моделирующее устройство может легко играть роль несбоящих процессоров. В раунде 1 противник просматривает не более t долей реального входа несбоящих процессоров. В соответствии со свойствами схемы разделения секрета Шамира, эти доли полностью случайны и, таким образом, могут моделироваться даже без знания реального входа (как и в случае с моделирующим устройством). В раунде 9 реальная доля распространена "скрытым образом" как случайный "мусор", а это позволит моделирующему устройству распространять сообщение несбоящих процессоров с необходимым распределением даже без знания реального входа.

С доказательством лемм 2.2 - 2.5 доказательство теоремы 2.3 следует считать законченным.

Здесь уместно сделать следующее замечание. Схемы (протоколы) конфиденциальных вычислений являются чрезвычайно сложным объектом исследований. Как видно из сказанного выше, даже описание и доказательство безопасности одного из базовых примитивов в протоколах конфиденциального вычисления функции, - схемы проверяемого разделения секрета являются чрезвычайно громоздкими и сложными. Демонстрация собственно протоколов конфиденциальных вычислений, решающих те или иные теоретические или прикладные задачи, выходит за рамки настоящей работы.

RL-прототип модели синхронных конфиденциальных вычислений

Зададимся вопросом "Существует ли реальный вычислительный аналог представленной модели синхронных конфиденциальных вычислений ?". Такой вопрос важен и с прикладной, и с содержательной точки зрения. Рассмотрим многопроцессорную супер ЭВМ семейства S-1 на базе сверхбыстродействующих процессоров MARK IIA (MARK V). Такую вычислительную систему назовем RL-прототипом (real-life) модели синхронных конфиденциальных вычислений в реальном сценарии.

Проект семейства систем высокой производительности для военных и научных применений (семейства S-1) является в США частью общей программы развития передовых направлений в области числовой обработки информации. Исходя из целей программы, можно сделать вывод о возможности применения указанной вычислительной системы в автоматизированных системах критических приложений. Поэтому требования высокой надежности и безопасности программного обеспечения систем являются обязательными.

В указанной многопроцессорной системе используются специально разработанные однопроцессорные машины, образующие семейство, то есть имеющие однотипную архитектуру. В систему может входить до 16 однопроцессорных ЭВМ, сравнимых по производительности с наиболее мощными из существующих супер ЭВМ. В дополнение к обычному универсальному оборудованию процессоры семейства S-1 оснащены специализированными устройствами, позволяющими выполнять высокоскоростные вычисления в тех прикладных областях, где планируется использование данной многопроцессорной системы. К таким операциям относятся вычисления функций синуса, косинуса, возведения в степень, логарифмирования, операции быстрого преобразования Фурье и фильтрации, операции умножения над матрицами и транспонирования.

Системы семейства S-1 предоставляют в распоряжение пользователя большую сегментированную память с виртуальной адресацией в несколько миллиардов 9-разрядных байтов. Процессоры соединены между собой с помощью матричного переключателя, который обеспечивает прямую связь каждого процессора с каждым блоком памяти (см. рис.2.10). При обращениях к той или иной ячейке памяти процессоры всегда получают последнее записанное в ней значение. Команды выполняются в последовательности: "чтение - операция - запись". С целью повышения быстродействия памяти в составе каждого процессора имеются две кэш-памяти: первая - объемом 64 Кбайт для хранения данных, вторая - объемом 16 Кбайт (с перспективой наращивания). В обоих типах кэш-памяти длина набора составляет 4, а длина строки 64 байта.

В операционной системе Amber, предназначенной для вычислительных систем семейства S-1, предусмотрена программа планировщик, который на нижнем уровне архитектуры системы обеспечивает эффективный механизм оперативного планирования заданий на одном процессоре. Основным правилом планирования здесь является назначения в порядке очереди. На уровне такого одноприоритетного планирования каждое задание выполняется до тех пор, пока не возникает необходимость ожидания какого-либо внешнего события или не истечет выделенный квант процессорного времени. Планировщик высокого уровня осуществляет более сложное планирование, в основу которого положена некоторая общая стратегия. Результатом его работы являются соответствующим образом измененные параметры планировщика нижнего уровня: приоритеты заданий или размеры квантов времени.

Одной из важнейших особенностей многопроцессорной системы семейства S-1 является наличие общей памяти большой емкости, позволяющей осуществлять широкомасштабный обмен данными между заданиями. Если два задания работают с одним и тем же сегментом памяти, они пользуются его единственной физической копией, в то время как другие области пространства адресов остаются в их собственном владении. Таким образом, задания оказываются защищенными от неосторожных попыток изменить их внутренне состояние. Между двумя заданиями можно установить жесткий режим чтения-записи, когда одному заданию разрешаются операции чтения-записи, а другому только операции чтения из данного сегмента.

Операционная система Amber имеет большие возможности для реконфигурации системы в случае возникновения сбоев (внештатных ситуаций). Если требуется вывести из конфигурации процессор, выполнение на нем приостанавливается и производится их перераспределение на другие процессоры. Когда процессор или память вводятся в рабочую конфигурацию, они становятся обычными ресурсами системы и загружаются по мере необходимости.

Таким образом, можно проследить широкий круг аналогий между моделью конфиденциальных вычислений и ее вычислительным прототипом. В этом случае центральный коммутатор совместно с устройствами памяти можно рассматривать и как широковещательный канал, и как набор конфиденциальных каналов связи между процессорами. Конфиденциальность каналов может рассматриваться в том случае, если существует возможность предотвратить несанкционированный доступ к блокам памяти или хранить и передавать данные в зашифрованном виде. Привязка во времени многопроцессорной системы S-1 осуществляется посредством устройства синхронизации. Параметром безопасности системы может являться длина строки (64-256 Кбайт). Вычислительная система типа S-1 позволяет осуществлять разработку прототипов распределенных вычислительных систем и исследование характеристик многосторонних протоколов различного типа, как криптографического характера, так и нет. К такой разновидности распределенных вычислений можно отнести следующие протоколы, имеющие, в том числе, и прикладное значение.

1. Протоколы византийского соглашения (определение дано выше).
2. Протоколы разделения секрета (определение дано выше).
3. Протоколы электронного голосования. В таком протоколе должно быть обеспечено три основных условия:
   * обеспечения правильности подачи и подсчета голосов;
   * обеспечения тайного голосования избирателей;
   * обеспечения невозможности срыва выборов или искажения их результатов.
4. Протоколы выработки общей случайной строки. Протокол позволяет безопасным образом группе участников, часть из которых являются нечестными, выработать с равномерным распределением вероятностей общую для всех случайную строку. Такой протокол является часто используемым вычислительным примитивом для построения более сложных протоколов распределенных вычислений.

Методы конфиденциальных вычислений могут позволить для многопроцессорных систем проектировать высокозащищенную программно-аппаратную среду для использования в автоматизированных системах критически уязвимых объектов информатизации.

###### 2.4.4. Криптопрограммирование

Криптопрограммирование посредством использования инкрементальных алгоритмов

Одним из основных инструментов методологии криптопрограммирования являются инкрементальные алгоритмы. Цель инкрементальной криптографии заключается в разработке криптографических алгоритмов обработки электронных данных, обладающих следующим принципиальным свойством. Если алгоритм применяется к электронным данным D для достижения каких-либо их защитных свойств, то применение инкрементального алгоритма к данным D, подвергнувшихся модификации - D`, должно осуществляться быстрее, чем необходимость заново обработать данный документ. В тех приложениях, когда указанные алгоритмы являются, например, алгоритмами шифрования электронных документов или их цифровой подписи, требование повышения эффективности инкрементальных алгоритмов является крайне желательным. Один из основных методов применения инкрементальных алгоритмов заключается в использовании их аутентификационных признаков для антивирусной защиты.

При обработке электронных документов инкрементальными алгоритмами рассматриваются такие операции обработки данных как "вставка" и "стирание" для символьных строк или "cut" - "вырезание и помещение в буфер" и "paste" - "извлечение из буфера и вставка" для текста. Основная задача здесь заключается в разработке эффективных инкрементальных алгоритмов для схем цифровой подписи и схем аутентификации сообщений, поддерживающих вышеупомянутые операции по модификации электронных данных. Такие алгоритмы должны обладать основным качественным свойством, а именно свойством "защиты от вмешательства", что, таким образом, и делает их применимыми для защиты программ от вирусов и других разрушающих программных средств. Основные криптографические примитивы, такие как шифрование и цифровая подпись имеют фундаментальную теоретическую базу. Во многих работах (см., например, обзор в работе [[28](http://citforum.ru/security/articles/kazarin/4.shtml#28)]) были даны базовые определения их криптографической стойкости, основанные на обобщенных теоретико-сложностных и теоретико-информационных предположениях. Главная проблема, которая остается и затрудняет использование на практике многих доказуемо стойких теоретических криптоконструкций, заключается в их пространственно-временной неэффективности. Инкрементальность, в этом смысле, является новой мерой эффективности, которая является вполне приемлемой во многих алгоритмических приложениях.

Пусть далее рассматривается процессор, защищенный от физического вмешательства, который имеет ограниченное количество безопасной локальной памяти. Необходимо получить доступ к файлам, находящихся на удаленных (возможно небезопасных) носителях, например, хост-станциях или www-серверах. Компьютерный вирус может атаковать удаленную станцию, и исследовать и изменять содержание удаленной информационной среды (но при этом он не имеет доступа к безопасной локальной памяти процессора). Для защиты файлов от таких вирусов, процессор вычисляет для каждого файла аутентификационный признак, как функцию от самого файла и ключа, который хранится в безопасной локальной памяти.

Внедрение вируса в файл не позволяет ему заново вычислять признак, и при реализации процесса верификации признака, таким образом, обнаружится вторжение вируса. Следует обратить внимание на то, что для корректной верификации аутентификационного признака защищенный процессор должен заново подтвердить подлинность файлов. Ясно, что наиболее привлекательным способом является модернизация аутентификационного признака быстрее, чем необходимость его вычисления заново. Эта проблема особенно сложна в том случае, когда локальная память не достаточно большая для того, чтобы хранить (даже временно) фрагмент файла или когда "слишком дорого" ввести в локальную память полный файл.

Идея инкрементальных алгоритмов, состоит в том, чтобы воспользоваться какими-либо имеющимися преимуществами такой организации программно-аппаратного взаимодействия и найти способы криптографических преобразований над электронными данными D не заново, а, скорее, как получение быстродействующей функции от значений криптографических преобразований над данными из которыхD был получен. Когда "изменения" не велики, инкрементальные методы могут дать большие преимущества по эффективности.

Основные элементы инкрементальной криптографии

Примитивы. Инкрементальность можно рассматривать для любого криптографического примитива. В данном случае рассматриваются два из них - цифровая подпись и шифрование. Инкрементальность далее рассматривается, как правило, для "прямых" преобразованиях, а именно для генерации подписи и шифровании, но все рассуждения будут верны и для "сопряженных" преобразований, а именно для верификации подписи и дешифрования.

Операторы модификации.Рассмотрим проблему модификации защищаемого файла в терминах применения фиксированного набора основных операций по модификации электронного документа. Например: замена блока в документе другим; вставка нового блока; удаление старого блока. Операции должны быть достаточно "мощны" для демонстрации реальных модификаций таких как: замена, вставка и удаление. Соответственно также рассматриваются операции "cut" и "paste", например, операции разбиения отдельного документа на два, а затем, вставка двух документов в один.

Инкрементальные алгоритмы. Зафиксируем базовое криптографическое преобразование **T**(например, цифровая подпись документа с некоторым ключом). Каждой элементарной операции модификации текста (например, вставки) будет соответствовать инкрементальный алгоритм ***I***. На вход этого алгоритма подаются: исходный файл; значения преобразования ***T*** на нем; описание модификаций; и возможно соответствующие ключи или другие параметры. Это позволит вычислить значение ***T*** для результирующего файла. Основная проблема здесь заключается в проектировании схем обработки файлов, обладающих эффективными инкрементальными алгоритмами. Предположим, что имеется подпись σold для файла Dold и файл D'old, измененный посредством вставки в файл Dold некоторых данных. Необходимо получить подпись путем подписывания строки, состоящей из σold и описания модификаций над документом Dold. Это схема называется схемой, зависящей от истории. Могут иметься приложения, когда такие действия являются приемлемыми. В большинстве же случаев это не желательно, так как делается большое количество изменений и затраты на верификацию подписи пропорциональны числу изменений. В связи с этим размеры подписи растут со временем. Чтобы избежать таких затрат необходимо использовать схемы, свободные от истории или HF-схемы. Все нижеприведенные схемы являются схемами, свободными от истории.

Безопасность. Свойство инкрементальности вводит новые проблемы безопасности и вызывает необходимость новых определений. Рассмотрим случай схем подписи и аутентификации. Разумно предположить, что противник не только имеет доступ к предыдущим подписанным версиям фалов, но также способен выдавать команды на модификацию текста в существующих файлах и получать инкрементальные подписи для измененных файлов. Такая атака на основе выбранного сообщения для инкрементальных алгоритмов подписи может вести к "взлому" используемой схемы подписи, которая не может быть взломана при проведении противником других атак, в тех случаях, когда не используются инкрементальные алгоритмы. Кроме того, в некоторых сценариях, например, при вирусных атаках можно предположить, что противник вмешивается в содержание существующих документов и аутентификационных признаков, полученных посредством схем подписи/аутентификации. Соответственно рассматриваются два определения безопасности: базовое и более сильное понятие безопасности, когда доказывается стойкость защиты от вмешательств.

Секретность инкрементальных схем. Исходя из вышесказанного, появляется новая проблема, которая проявляется в инкрементальном сценарии, а именно - проблема секретности различных версий файлов. Предположим μ - подпись кода М и μ' является подписью слегка измененного кода M'. Тогда, наилучшим вариантом было бы получить такую подпись μ', которая давала бы как можно меньше информации, об оригинальном коде М.

Метод защиты файлов программ посредством инкрементального алгоритма маркирования

Основные определения. Пусть АУТ(m) - обычный алгоритм аутентификации сообщений и АУТa(m)- функция маркирования сообщения m, индуцированная схемой АУТ с ключом аутентификации a. Пусть ВЕРa(m,β) - соответствующий алгоритм верификации, где β={true, false} - предикат корректности проверки.

Далее будут использоваться деревья поиска и, следовательно, необходимо напомнить, что 2-3-дерево имеет все концевые узлы (листья) на одном и том же самом уровне/высоте (как и в случае сбалансированных двоичных деревьев) и каждая внутренняя вершина имеет или 2, или З дочерних узла [[2](http://citforum.ru/security/articles/kazarin/4.shtml#2)]. В данном случае 2-3-дерево подобно двоичному дереву является упорядоченным деревом и, таким образом, концевые узлы являются упорядоченными. Пусть Vh - определяет множество всех строк длины не больше h, ассоциированных очевидным способом с вершинами сбалансированного 2-3-дерева высотыh. Маркированное дерево может рассматриваться как функция Т: Vh - >{0,1}\*, которая приписывает аутентификационный признак (АП) каждой вершине.

Пусть совокупный аутентификационный признак файла F получен посредством использования 2-З-дерева аутентификационных признаков каждого из блоков файла F=F[1],...,F[l] (далее такое дерево будет называться маркированным деревом). Каждая вершина w ассоциирована с меткой, которая состоит из АП (аутентифицирующих дочерние узлы) и счетчика, представляющего число узлов в поддереве с корнем w.

Алгоритм маркирования. Алгоритм создания 2-3-дерева аутентификационных признаков (алгоритм маркирования) работает следующим образом:

* для каждого i, пусть T(w)=АУТα(F[i]), где w - i-тый концевой узел;
* для каждого не - концевого узла w, пусть Т(w)=АУТα((L1,L2,L3),рзм), где Li - метка i-того дочернего узла w(в случае, если w имеет только два дочерних узла, то L3=γ) и рзм - число узлов в поддереве с корнем w);
* для корня дерева Т(λ)=АУТα((L1,L2,L3),Id,счт), где Id- название документа и счт - соответствующее показание счетчика (связанное с этим документом).

Инкрементальный алгоритм маркирования. Предположим, что файл F, аутентифицированный маркированным деревом, подвергается операции замены, то есть j-тый блок файла F заменен блоком F(σ). Сначала необходимо проверить, что путь от требуемого текущего значения до корня дерева корректен. Для этого необходимо выполнить следующий алгоритм.

Пусть u0,...uh - путь из корня u0=λ к j-тому концевому узлу обозначается как uh. Тогда:

* проверить, что ВЕРα принимает Т(λ) как корректный АП строки Т(λ)=АУТa((L1,L2,L3),Id,счт), где Id- название документа и счт - текущее значение счетчика (связанного с этим документом);
* для i=1,...,h-1 проверить, что ВЕРα принимает Т(ui) как корректный АП строки ((L1,L2,L3),рзм), гдеLi - метка i-того дочернего узла w (в случае, еслиw имеет только два дочерних узла, то L3=γ) и рзм - число узлов в поддереве с корнем w));
* проверить, что ВЕРα принимает Т(uh) как корректный АП блока F[j]

.

Если все эти проверки успешны, тогда совокупный АП файла F получается следующим образом:

* установить Т(uh):=АУТ(F(σ));
* для i=h-1,...,1 установить Т(ui):=АУТ(Т(ui1),Т(ui2),Т(ui3));
* установить Т(λ):=АУТ((Т(ui0),Т(ui1),Т(ui1)),Id,счт+1).

Следует подчеркнуть, что значения Т на всех других вершинах (то есть, не стоящих на пути u0,...,uh)остаются неизменяемыми.

Следует отметить, что предлагаемая инкрементальная схема маркирования имеет дополнительное свойство, заключающееся в том, что она безопасна даже для противника который может "видеть" как отдельные аутентификационные признаки, так и все маркированное дерево и может даже "вмешиваться" в эти признаки. Для каждого файла, пользователь должен хранить в локальной безопасной памяти ключ x схемы подписи, имя файла и текущее значение счетчика. Всякий раз, когда пользователь хочет проверить целостность файла, он проверяет корректность маркированного дерева открытым образом.

Наиболее эффективным является использование инкрементального алгоритма маркирования для защиты программ, использующих постоянно обновляющие структуры данных, например, файл с исходными данными или итерационно изменяемыми переменными.

###### 2.4.5. Защита программ и забывающее моделирование на RAM-машинах

Основные положения

В этом разделе рассматриваются теоретические аспекты защиты программ от копирования. Эта задача защиты сводится к задаче эффективного моделирования RAM-машины (машины с произвольным доступом к памяти) посредством забывающей RAM-машины. Следует заметить, что основные результаты по данной тематике (их можно получить на соответствующем личном интернетовском сайте) принадлежат О. Голдрайху и Р. Островски и эти исследования активно продолжаются в настоящее время.

Машина является забывающей, если последовательность операций доступа к ячейкам памяти эквивалентна для любых двух входов с одним и тем же временем выполнения. Например, забывающая машина Тьюринга - это машина, для которой перемещение головок по лентам является идентичным для каждого вычисления и, таким образом, перемещения не зависят от фактического входа.

Необходимо выделить следующую формулировку ключевой задачи изучения программы по особенностям ее работы. "Как можно эффективно моделировать независимую RAM-программу на вероятностной забывающейRAM-машине". В предположении, что односторонние функции существуют, далее показывается, как можно сделать некоторую схему защиты программ стойкой против полиномиально-временного противника, которому позволено изменять содержимое памяти в процессе выполнения программы в динамическом режиме.

Центральный процессор, имитирующий взаимодействие. Неформально, будем говорить, что центральный процессор имитирует взаимодействие с соответствующими зашифрованными программами, если никакой вероятностный полиномиально-временной противник не может различить следующие два случая, когда по данной зашифрованной программе как входу:

* противник экспериментирует с оригинальным защищенным ЦП, который пытается выполнить зашифрованную программу, используя память;
* противник экспериментирует с "поддельным" (фальсифицированным) ЦП. Взаимодействия поддельного ЦП с памятью почти идентичны тем, которые оригинальный ЦП имел бы с памятью при выполнении фиксированной фиктивной программы. Выполнение фиктивной программы не зависит по времени от числа шагов реальной программы. Не зависимо от времени поддельный ЦП (некоторым "волшебным" образом) записывает в память тот же самый выход, который подлинный ЦП написал, выполняя "реальную" программу.

При создании ЦП, который имитирует эксперименты, имеются две проблемы. Первая заключается в том, что необходимо скрывать от противника значения, сохраненные и восстановленные из памяти, и предотвращать попытки противника изменять эти значения. Это делается с использованием традиционных криптографических методов (например, методов вероятностного шифрования и аутентификации сообщений). Вторая проблема заключается в необходимости скрывать от противника последовательность адресов, к которым осуществляется доступ в процессе выполнения программы (здесь и далее это определяется как сокрытие модели доступа).

Сокрытие оригинальной модели доступа к памяти - это абсолютно новая проблема и традиционные криптографические методы здесь не применимы. Цель в таком случае состоит в том, чтобы сделать невозможным для противника получить о программе что-либо полезное из модели доступа. В этом случае ЦП не будет выполнять программу обычным способом, однако он заменяет каждый оригинальный цикл "выборки/запоминания" многими циклами "выборки/запоминания". Это будет "путать" противника и предупреждать его от "изучения" оригинальной последовательности операций доступа к памяти (в отличие от фактической последовательности). Следовательно, противник не может улучшить свои способности по восстановлению оригинальной программы.

Ценой, которую необходимо заплатить за защиту программ, таким образом, является быстродействие вычислений. Неформально говоря, затраты на защиту программ определяются как отношение числа шагов выполнения защищенной программы к числу шагов исходного кода программы. Далее показывается, что эти затраты полиномиально связаны с параметром безопасности односторонней функции, что подтверждается следующим тезисом.

Предположим, что односторонние функции существуют и пусть k - параметр безопасности функции. Тогда существует эффективный способ преобразования программ в пары, состоящие из физически защищенного ЦП с k битами внутренней защищенной памяти и соответствующей зашифрованной программы такой, что ЦП имитирует взаимодействие с зашифрованной программой, реализуемое за время, ограниченное poly(k). Кроме того, t команд оригинальной программы выполняются с использованием менее чем за tk0(1) команд зашифрованной программы, а увеличение размера внешней памяти ограничиваются коэффициентом k.

Вышеупомянутый результат доказывается посредством сведения задачи создания ЦП, который нарушает эксперименты по вмешательству, к задаче сокрытия модели доступа. По-существу, последняя задача формулируется как задача моделирования на независимой забывающей RAM-машине (см. ниже).

Моделирование на забывающих RAM-машинах

Для каждой приемлемой модели вычислений существует преобразование независимых машин в эквивалентные забывающие машины (т.е., в забывающие машины, вычисляющие ту же самую функцию). Вопрос заключается в ресурсозатратах для этих преобразований, а именно в определении времени замедления работы забывающей машины. Например, машина Тьюринга с одной лентой может моделироваться посредством забывающей машины Тьюринга с двумя лентами с логарифмическим замедлением времени выполнения. Ниже исследуется подобный процесс, но для модели вычислений с произвольным доступом к памяти (RAM-машины). Основное достоинство RAM-машины - это способность мгновенно получать доступ к произвольным ячейкам памяти. В контексте настоящей работы, приводится следующий основной неформальный результат для RAM-машины.

Пусть RAM(m) означает RAM-машину с m ячейками памяти и доступом к случайному оракулу. Тогда t шагов независимой RAM(m)-программы может моделироваться за менее чем O(t(log2 t)3) шагов на забывающей RAM(m(log2 m)2).

Таким образом, можно увидеть, как провести моделирование независимой RAM-программы на забывающей RAM-машине с полилогарифмическим (от размера памяти) замедлением времени выполнения. В то же время, простой комбинаторный аргумент показывает, что любое забывающее моделирование независимой RAM-машины должно иметь среднее число Ω(lоgt) затрат. В связи с эти приводится следующий аргумент.

Пусть машина RAM(m) - определена как показано выше. Тогда каждое забывающее моделирование RAM(m)-машины должно содержать не менее max{m,(t-1)log m} операций доступа, чтобы смоделировать  tшагов оригинальной программы.

Далее рассмотрим сценария наихудшего случая, в котором наблюдатель (или в данном случае противник) активно пытается получить информацию, вмешиваясь в процесс вычислений. Вопрос заключается в том, можно ли гарантировать, что воздействие противника является забывающим по входу. Неформально говоря, моделирование RAM-машины на забывающей RAM-машине является доказуемо защищенным от вмешательства, если моделирование остается забывающим (т.е. не вскрывает какой-либо информации о входе за исключением его длины) даже в случае, когда независимый "мощный" противник исследует и изменяет содержимое памяти. В связи с этим приводится следующий аргумент.

В условиях определения RAM(m)-машины t шагов независимой RAM(m)-программы могут быть промоделированы (доказуемо защищенным от вмешательства способом) менее чем за O(t(log2t)3) шагов на забывающей машине RAM(m(log2m)2).

Необходимо отметить, что вышеприведенные результаты относятся к RAM-машинам с доступом к случайному оракулу. Чтобы получить результаты для более реалистичных моделей вероятностных RAM-машин, необходимо заменить случайный оракул, используемый выше, псевдослучайной функцией. Такие функции существуют в предположении существования односторонних функций с использованием короткого действительно случайно выбранного начального значения.

Модели и определения

Далее рассматривается модель RAM как пара интерактивных машин с ограниченными ресурсами, и даются два базовых понятия: понятие защиты программ и понятие моделирования на забывающей RAM-машине.

RAM-машина как пара интерактивных машин. В данном разделе RAM-машина представляется как две интерактивные машины: центральный процессор (ЦП) и модуль памяти (МП). Задача исследований сводится изучению взаимодействия между этими машинами. Для лучшего понимания необходимо начать с определения интерактивной машины Тьюринга.

Интерактивная машина Тьюринга - многоленточная машина Тьюринга, имеющая следующие ленты:

* входная лента "только-для-чтения";
* *выходная лента "только-для-записи";*
* *рабочая лента "для-записи-и-для-чтения";*
* *коммуникационная лента "только-для-чтения";*
* коммуникационная лента "только-для-записи"

.

Под ITM(c,w) обозначается машина Тьюринга с рабочей лентой длины w и коммуникационными лентами, разделенными на блоки c-битной длины, которая функционирует следующим образом. Работа ITM(c,w) на входе у начинается с копирования у в первые |y| ячеек ее рабочей ленты. В случае если |y|>w, выполнение приостанавливается немедленно. В начале каждого раунда, машина читает следующий c-битный блок с коммуникационной ленты "только-для-чтения". После некоторого внутреннего вычисления, использующего рабочую ленту, раунд завершен с записью с битов на коммуникационную ленту "только-для-записи". Работа машины может завершиться в некоторой точке с копированием префикса ее рабочей ленты на выходную ленту машины. Теперь можно определить ЦП и МП как интерактивные машины Тьюринга, которые взаимодействуют друг с другом, а также можно ассоциировать коммуникационную ленту "только-для-чтения" ЦП с коммуникационной лентой "только-для-записи" МП и наоборот. Кроме того, и ЦП, и МП будут иметь ту же самую длину сообщений, то есть, параметр с, определенный выше. МП будет иметь рабочую ленту размером, экспоненциальным от длины сообщений, в то время как ЦП будет иметь рабочую ленту размером, линейным от длины сообщений. Каждое сообщение может содержать "адрес" на рабочей ленте МП и/или содержимое регистров ЦП.

Далее используем k как параметр, определяющий и длину сообщений, и размер рабочих лент ЦП и МП. Кроме того, длина сообщений будет равна k+2+k', а размер рабочей ленты будет равен 2kk', где k'=O(k).

Для каждого k*http://citforum.ru/security/articles/kazarin/e.gif****N*** определим MEMk как машину IТМ(k+2+O(k),2kO(k)), работающую точно так, как определено выше. Рабочая лента разбивается на 2k слов, каждое размером O(k). После копирования входа на рабочую ленту машина MEMk является машиной, управляемой сообщениями. После получения сообщения (i,a,v), где ihttp://citforum.ru/security/articles/kazarin/e.gif{0,1}2http://citforum.ru/security/articles/kazarin/3r.gif{"запомнить","выборка","стоп"}, ahttp://citforum.ru/security/articles/kazarin/e.gif{0,1}k и vhttp://citforum.ru/security/articles/kazarin/e.gif{0,1}O(k), машина MEMkработает следующим образом.

Если i="запоминание", тогда машина MEMk копирует значение v из текущего сообщения в число а рабочей ленты.

Если i="выборка", тогда машина MEMkпосылает сообщение, состоящее из текущего числа а (рабочей ленты).

Если i="стоп", тогда машину MEMk копирует префикс рабочей ленты (как специальный символ) на выходную ленту и останавливается.

Далее, пусть для каждого k*http://citforum.ru/security/articles/kazarin/e.gif****N*** определим CPUk как машину IТМ(k+2+O(k),O(k)), работающую точно так, как определено выше. После копирования входа на свою рабочую ленту, машина CPUk выполняет вычисления за время, ограниченное poly(k), используя рабочую ленту, и посылает сообщение, определенное в этих вычислениях. В следующих раундах CPUk - является машиной, управляемой сообщениями. После получения нового сообщения машина CPUk копирует сообщение на рабочую ленту и, основываясь на вычислениях на рабочей ленте, посылает свое сообщение. Число шагов каждого вычисления на рабочей ленте ограничено фиксированным полиномом от k.

Единственная роль входа ЦП должна заключаться в инициализации регистров ЦП, и этот вход может игнорироваться при последовательном обращении. "Внутреннее" вычисление ЦП в каждом раунде соответствует элементарным операциям над регистрами. Следовательно, число шагов, принимаемых в каждом таком вычислении является фиксированным полиномом от длины регистра (которая равна O(k)). Теперь можно определить RAM-модель вычислений, как совокупность RAMk-машин для каждого k.

Для каждого k*http://citforum.ru/security/articles/kazarin/e.gif****N*** определим машину RAMk как пару (CPUk, MEMk), где ленты "только-для-чтения" машины CPUk совпадают с лентами "только для записи" машины MEMk, а ленты "только-для-записи" машины CPUk совпадают с лентами "только-для-чтения" машины MEMk. Вход RAMk - это пара (s,y), где s- вход (инициализация) для CPUk и у - вход для MEMk. Выход машины RAMk по входу (s,у), обозначаемый как RAMk(s,у), определен как выход MEMk(y) при взаимодействии с CPUk(s).

Для того, чтобы рассматривать RAM-машину как универсальную машину, необходимо разделить вход у машины MEMk на "программу" и "данные". То есть, вход у памяти разделен (специальным символом) на две части, названные программой (обозначенной П) и данными (обозначаемыми x).

Пусть RAMk и s фиксированы и у=(П,х). Определим выход программы П на входных данных х, обозначаемый через П(x) как RAMk(s,у). Определим время выполнения П на данных х, обозначаемое через tП(x), как сумму величины (|у|+|П(x)|) и числа раундов вычисления RAMk(s,у). Определим также размер памяти программы П для данных х, обозначаемый через sП(x) как сумму величины |у| и числа различных адресов, появляющихся в сообщениях, посланных CPUk к MEMk в течение работы RАМk(s,у).

Легко увидеть, что вышеупомянутая формализация непосредственно соответствует модели вычислений с произвольным доступом к памяти. Следовательно, "выполнение П на х" соответствует раундам обмена сообщениями при вычислении RAMk(**'**,(П,х)). Дополнительный член |y| + |П(x)| в tП(x) поясняет время, потраченное при чтении входа и записи выхода, в то время как каждый раунд обмена сообщениями представля-ет собой единственный цикл в традиционной RAM-модели. Член |y| в sП(х) объясняет начальное пространство, взятое по входу.

Дополнения к базовой модели и вероятностные RAM-машины. Приводимые ниже результаты верны дляRAM-машин, которые являются вероятностными в очень строгом смысле. А именно ЦП в этих машинах имеет доступ к случайным оракулам. Однако в предположении существования односторонних функций, случайные оракулы могут быть эффективно реализованы посредством псевдослучайных функций.

Для каждогоkhttp://citforum.ru/security/articles/kazarin/e.gif N определим оракульный CPUk как CPUk с двумя дополнительными лентами, названными оракульными лентами. Одна из этих лент является "только-для-чтения", а другая "только-для-записи". Всякий раз, когда машина входит в специальное состояние вызова оракула, содержимое оракульной ленты "только-для-чтения" изменяется мгновенно (т.е., за единственный шаг) и машина переходит к другому специальному состоянию. Строка, записанная на оракульной ленте "только-для-чтения" между двумя вызовами оракула называется запросом, соответствующим последнему вызову. Будем считать, что CPUk имеет доступ к функции f, если делается запрос q и оракул отвечает и изменяет содержимое оракульной ленты "только-для-чтения" на f(q). Вероятностная машина CPUk - это оракульная машина CPUkс доступом к однородно выбранной функции

f:{0,1}O(k) - > {0,1}.

Для каждого k http://citforum.ru/security/articles/kazarin/e.gif N определим оракульную RAMk-машину как RAMk-машину, в которой машина CPUkзаменена на оракульную CPUk. Скажем, что эта RAMk-машина имеет доступ к функции f, если CPUkимеет доступ к функции f и будем обозначать как RAMkf . Вероятностная RAMk-машина - это RAMk-машина, в которой CPUk заменен вероятностным CPUk. (Други-ми словами, вероятностная RAMk-машина - это оракульная RAMk-машина с доступом к однородно выбранной функции).

Повторные выполненияRAM-машины. Для решения проблемы защи-ты программ необходимо использовать повторные выполнения "одной и той же" RAM-программы на нескольких входах. Задача состоит в том, что RАМ-машина начинает следующее выполнение с рабочими лентами ЦП и МП, имеющих содержимое, идентичное их содержимому при окончании предыдущего выполнения программы.

Для каждого k http://citforum.ru/security/articles/kazarin/e.gif N , повторные выполнения RAMk-машины на входной последовательности y1,y2,...рассматриваются как последовательность вычислений RAMk-машины, при котором первое вычисление началось с входа y1, когда рабочие ленты и СРUk, и MEMk пусты и i-тое вычисление начинается с входауi, когда рабочая лента каждой машины (т.е., и СРUk, и MEMk) содержит ту же самую строку, которую она содержала по окончании i-1 вычисления.

Эксперименты с RAM-машиной. Рассматриваются два типа противников. Оба могут неоднократно инициировать работу RAМ-машины на входах по своему выбору. Различия между двумя типами противников состоит в их способности модифицировать коммуникационные ленты ЦП и МП в процессе вычислений. Вмешивающемуся противнику позволено как читать, так и записывать на эти ленты свою информацию (т.е., просматривать и изменять содержание взаимодействия), в то время как невмешивающемуся противнику позволено только читать эти ленты (то есть, только просматривать сообщения). В любом случае не надо позволять противнику иметь те же самые права доступа к рабочей ленте МП, так как содержимое этой ленты полностью определено начальным входом и сообщениями, посланными ЦП. Кроме того, в обоих случаях противник не имеет никакого доступа к внутренним лентам ЦП (т.е., к рабочим и оракульным лентам ЦП).

Для простоты, основное внимание будет уделяться противникам с экспоненциально ограниченным временем выполнения. Кроме того, время выполнения противника ограничено сверху 2n, где n - размер рабочей ленты ЦП. На практике противник будет ограничен по времени некоторым полиномом от длины рабочей ленты ЦП.

Невмешивающийся противник, обозначаемый как ADV является вероятностной машиной, которая на входе k и "шифрованной программе" α, которая имеет следующий доступ к оракульной RAMk-машине. Машина ADV инициирует повторные выполнения RAMk-машины на входах по своему выбору до тех пор, пока общее время выполнения не стане равным 2k. В процессе каждого из этих выполнений, машина ADVимеет доступ "только-для-чтения" к коммуникационным лентам между CPUkи MEMk.

Вмешивающийся противник определяется аналогично невмешивающемуся противнику за исключением того, что в процессе повторных вы-полнений противник имеет доступ для чтения и записи к коммуникацион-ным лентам между CPUk и МЕМk.

Преобразования, защищающие программное обеспечение

Определим компиляторы, которые по данной программе П, произво-дят пару (f,Пf), где f - случайно выбранная функция и Пf - "зашифрованная программа", которая соответствует П и f. Здесь имеется в виду оракульная RAM-машина, которая на входе (Пf,х) и при доступе к оракулу f моделирует выполнение П на данных х так, чтобы это моделирование "защищало бы" оригинальную программу П.

Далее даются определения компиляторов как набора преобразований программ в программно-оракульные пары, которые при выполнении оракульных RAM-программ являются функционально эквивалентными выполнениям оригинальных программ.

Компилятор, обозначаемый через C, является вероятностным отображением, которое по входу целочисленного параметра k и программы П для RAMk возвращает пару (f,Пf) так, чтобы

* f:{0,1}O(k) - > {0,1} - случайно выбранная функция;
* |Пf| =O(|П|);

Для k'=k+O(log k) существует оракульная RAMk'-машина такая, что для каждой П, каждой f и каждого xhttp://citforum.ru/security/articles/kazarin/e.gif{0,1} инициируется RAMk' на входе (Пf,x) и при доступе к оракулу f обеспечивает П(x).

Оракульная RAMk'-машина отличается от RAMk-машины в том, что RAMk' имеет доступ к оракулу, в то время как RAMk нет. Понятно, что RAMk' имеет большую память, а именно RAMk'-машина состоит из 2k'=poly(k)2k слов, в то время как память RAMk состоит из 2k слов.

Компиляторы, как определено выше, преобразовывают детерминированные программы в "зашифрованные программы", которые выполняются на вероятностных RAM-машинах. Теперь непосредственно обратимся к определениям компилятора, защищающего программное обеспечение.

Оракул спецификации для программы П - это оракул, который на запрос x возвращает тройку (П(x),tП(x),sП(x)).

Отметим, что tП(x) и sП(x) обозначает время выполнения и пространственные размеры программы П на данных x. Далее даются основное определение для задачи защиты программ. В этом определении ADVможет рассматриваться как вмешивающийся, так и невмешивающийся противник.

Для данного компилятора C и противника ADV, компилятор C защищает программное обеспечение против противника ADV, если существует вероятностная оракульная (в стандартном смысле) машина М, удовлетворяющая следующим условиям:

* (М функционирует примерно за то же самое время, как и ADV): Существует полином p(**'**) такой, что для каждой строки α время выполнения М на входе (k', |α|) (и с учетом доступа к случайному оракулу) было ограничено p(k')T, где T обозначает время выполнения ADV при экспериментировании с RAMk' на входе α.
* (М с доступом к оракулу спецификации обеспечивает выход почти идентичный выходу ADVпосле экспериментирования с результатами работы компилятора): Для каждой программы П статистическое расстояние между следованием двух распределений вероятностей ограничено 2-k'.

Распределение выхода машины ADVпри экспериментировании с RAMkf на входе Пf, где (f,Пf)< - C(П). Отметим, что RAMkf обозначает ин-терактивную пару (CPUk',MEMk'), где CPUk' имеет доступ к оракулу f. Распределение берется над пространством вероятностей, состоящим из всех возможных выборов функции f и всех возможных результатов выработки случайного бита ("подбрасываний монеты") машины ADV с равномерным распределением вероятностей.

Распределение выхода оракульной машины М на входе (k',O( П )) и при доступе к оракулу спецификации для П. Распределение берется над пространством вероятностей состоящим из всех возможных результатов подбрасываний монеты машины М с равномерным распределением вероятностей.

Компилятор C обеспечивает слабую защиту программ, если C защищает программы против любого не вмешивающего противника. Компилятор C обеспечивает доказуемую защиту программ от вмешательства, если C защищает программы против любого вмешивающего противника.

Далее будет определяться затраты защиты программ. Необходимо напомнить, что для простоты, мы ограничиваем время выполнения программы П следующим условием: tП(x)> |П| + |x| для всех x.

Пусть C - компилятор и g:N - > N - некоторая целочисленная функция. Затраты компилятора C на большинстве аргументов g, если для каждой П, каждого xhttp://citforum.ru/security/articles/kazarin/e.gif {0,1}\* и каждой случайно выбранной fтребуемое время выполнения RAMk' на входе (Пf,x) и при доступе к оракулу f ограничены сверху g(T)T, где T=tП(x).

Определение забывающейRAM-машины и забывающего моделирования

Необходимо начать с определения модели доступа как последовательности ячеек памяти, к которым ЦП обращается в процессе вычислений. Это определение распространяется также на оракульный ЦП.

Модель доступа, обозначаемая как Аk(y) детерминированной RAMk-машины на входе y - это последовательность (a1,...,ai,...) такая, что для каждого i, i-тое сообщение, посланное CPUk при взаимодействии с MEMk(y) имеет форму (',ai,').

При рассмотрении вероятностных RAM-машин, мы определяем случайную величину, которая для каждой возможной функции f принимает модель доступа, соответствующая вычислениям, в которых RAM-машина имеет доступ к этой функции. В связи с этим дается следующее определение.

Модель доступа, обозначаемая как http://citforum.ru/security/articles/kazarin/49.gif(y) вероятностной RAMk-машины на входе y - это случайная величина, которая принимает значение модели доступа машины RAMk на некотором входе y и при доступе к однородно выбранной функции f.

Теперь можно перейти к определению забывающей RAM-машины. Мы определяем забывающую RAM-машину как вероятностную RAM-машину, для которой распределение вероятностей последовательности ад-ресов памяти, к которым осуществляется доступ в процессе выполнения программы, зависит только от времени выполнения и не зависит от конкретного частичного входа. Для каждого khttp://citforum.ru/security/articles/kazarin/e.gif N определим забывающую RAMk-машину как вероятностную RAMk-машину, удовлетворяющую следующему условию. Для каждых двух строк y1и y2, если |http://citforum.ru/security/articles/kazarin/49.gif(y1)| и |http://citforum.ru/security/articles/kazarin/49.gif(y2)| идентично распределены, тогда также идентично распределены http://citforum.ru/security/articles/kazarin/49.gif(y1) и http://citforum.ru/security/articles/kazarin/49.gif(y2) .

Интуитивно, последовательность операций доступа к памяти забывающей RAMk-машины не открывает никакой информации относительно входа за исключением значения времени выполнения на этом входе.

Определения RAM-машины и забывающей RAM-машины необходимо для того, чтобы дать точное определение забывающего моделирования независимой RAM-машины посредством забывающей RAM-машины. Определение моделирования в данном случае минимально необходимое, - требуется только, чтобы обе машины вычисляли одну и ту же функцию. Кроме того, необходимо потребовать, чтобы входы, имеющие идентичное время выполнения на оригинальной RAM-машине, сохраняли бы идентичное время выполнения на забывающей RAM-машине. Для простоты, ниже представляется только определение для забывающего моделирования детерминированных RAM-машин.

Для данных машин, - вероятностной RAM'k', и RAMk вероятностная машина RAM'k' моделирует забывающим образом RAMk, если выполняются следующие условия:

* вероятностная машина RAM'k' моделирует RAMk с вероятностью 1. Другими словами, для каждого входа y и каждого выбора оракульной функции f выход оракула RAM'k' на входе y и при доступе к оракулу f равняется выходу RAMk на входе y.
* вероятностная машина RAM'k' - является забывающей. Необходимо подчеркнуть, что здесь рассматривается модель доступа RAM'k' на фиксированном входе и случайно выбранной оракульной функции.

Случайная величина, представляющая собой время выполнения веро-ятностной RAM'k' на входе yполностью определена текущим временем RAMk на входе y.

Следовательно, модель доступа при забывающем моделировании (которая описывается случайной величиной, определенной над выбором случайного оракула) имеет распределение, зависящее только от времени выполнения оригинальной машины. А именно, пусть http://citforum.ru/security/articles/kazarin/49.gif(y) обозначает модель доступа при забывающем моделировании RAMk на входе y. Тогда http://citforum.ru/security/articles/kazarin/49.gif(y1) и http://citforum.ru/security/articles/kazarin/49.gif(y2) идентично распределены, если время выполнения RAMk на этих входах (т.е., y1и y2) идентично.

Теперь мы обратимся к определению затрат забывающего моделирования. Для данных вероятностных машин RAM'k' и RAMk предположим, что вероятностная RAM'k' моделирует забывающим образом вычисления RAMk и путь g:**N** - > **N** - есть некоторая целочисленная функция. Тогда затраты на моделирование являются не больше g, если для каждого y требуемое время выполнения RAM'k' на входе yограничено сверху g(T)T, где T обозначает время выполнения RAMk на входе y.

Моделирование с метками времени. В заключение этого подраздела приводится свойство некоторогоRAM-моделирования. Это свойство требует, чтобы при восстановлении значения из ячеек памяти, ЦП "знает" сколько раз содержимое этих ячеек модифицировалось. То есть, для данного адреса МП a и общего числа команд (обозначенных j), выполнение всех команд ЦП "запомнить в ячейку a" может быть эффективно вычислено алгоритмом Q(j,a). Далее рассматривается только моделирование детерминированных RAM-машин.

Для данной оракульной машины RAM'k', машины RAMk предположим, что оракульная RAM'k' с доступом к оракулу f' моделирует вычисления RAM'k'. Тогда моделирование является моделированием с метками времени, если существует O(k')-временной алгоритм Q(',') такой, что выполняется следующее условие. Пусть (i,a,v) - j-тое сообщение, посланное CPU'k' (в процессе повторных выполнений RAM'k'). Тогда, число предыдущих сообщений формы ("запомнить",a,"), посланных CPU'k', равняется точно (j,a). Далее, необходимо обратить запустить алгоритм на Q(j,a) для получения номер версии(a) в раунде j. Таким образом, чтобы "знать" номер версии любого адреса в некоторый момент времени, достаточно для ЦП сохранить счет числа шагов, которые выполняются. Подчеркнем, что ЦП не мог бы позволить себе хранить номер версии всех адресов памяти, так что проставление меток времени важно для получения доказуемой защиты программ от вмешательства.

Сведение защиты программ к забывающему моделированию на RAM-машине

Наконец сейчас будет показано, как свести задачу защиты программ к задаче моделирования RAM-машины на забывающей RAM-машине. Эта задача заключается в сокрытии модели доступа, полностью игнорируя тот факт, что содержимое памяти и коммуникаций между ЦП и МП доступно для противника. Мы начинаем со сведения задачи достижения слабой за-щиты программ (т.е., защита от не вмешивающихся противников) к построению забывающего RAM-моделирования. Далее мы сводим задачу защиты программ (доказуемой защиты от вмешательства) к построению забывающего моделирования с метками времени.

Защита программ от не вмешивающихся противников. Напомним, что противник называется не вмешивающимся, если все выбранные им входы инициируют выполнение программы на них и он читает содержимое памя-ти и коммуникаций между ЦП и МП при таком выполнении. Без потери общности, достаточно рассматривать противников, которые только читают коммуникационные ленты (так как содержимое ячеек памяти определено входом и коммуникациями между ЦП и МП). При использовании забы-вающего моделирования универсальной RAM-машины остается только скрыть содержимое "области значений" в сообщениях, обмениваемых ме-жду ЦП и МП. Это делается посредством шифрования, которое использует случайный оракул.

**Теорема 2.1.** Пусть {RAMk}khttp://citforum.ru/security/articles/kazarin/e.gifN - вероятностная RAM-машина, которая выполняет забывающее моделирование универсальной RAM-машины. Кроме того, предположим, что t шагов оригинальной RAM-машины моделируются за менее чем g(t)t шагов забывающей RAM-машины. Тогда существует компилятор, который защищает программы от невмешивающихся противников с затратами не болеO(g(t)).

**Доказательство.** Информация, доступная не вмешивающемуся противнику состоит из сообщений, обмениваемых между ЦП и МП. Напомним, что сообщения от CPUk к MEMk имеют форму (i,a,v), где ihttp://citforum.ru/security/articles/kazarin/e.gif{выборка, сохранить, стоп}, ahttp://citforum.ru/security/articles/kazarin/e.gif{1,2,...,2k} и vhttp://citforum.ru/security/articles/kazarin/e.gif{0,1}O(k), в то время как сообщения от MEMk до CPUkимеют форму vhttp://citforum.ru/security/articles/kazarin/e.gif{0,1}O(k). При забывающем моделировании, по определению, "область адресов" (т.е., a) не вскрывает никакой информации относительно входа y=(Пf,x). Просто необходимо устранить возможность, когда "область команд" (т.е., i) обеспечивает любую информацию посредством модификации ЦП так, чтобы всегда имелся доступ к ячейкам памяти при первой выборке. Следовательно, все что осталось должно "зашифровывать" содержимое области значений (т.е., v) так, чтобы ЦП мог восстанавливать оригинальное значение. Идея состоит в том, чтобы выполнить шифрование, используя оракул, доступный ЦП.

Для шифрования CPUk содержит специальный счетчик, обозначаемый счт, инициализированный нулевым значением. Мы модифицируем RAMk добавлением случайного оракула, обозначаемого f. Понятно, что новый случайный оракул может быть объединен со случайным оракулом, используемым при забывающем моделировании. Всякий раз, когда CPUk должен сохранять значение (либо старое значение, которое только читалось, либо новое значение) в памяти MEMk, счетчик счт увеличивается и значение vшифруется посредством пары (v*http://citforum.ru/security/articles/kazarin/plus.gif*f(счт),счт), где http://citforum.ru/security/articles/kazarin/plus.gif обозначает поразряд-ную операцию "исключающую или". При восстановлении пары (u,j), зашифрованное значение восстанавливается посредством вычисления u*http://citforum.ru/security/articles/kazarin/plus.gif*f(j). Подчеркнем, что и шифрование, и дешифрование может быть легко вы-полнены, когда имеется доступ к оракулу f.

Компилятор C, защищающий программное обеспечение, функционирует следующим образом. На входе параметр k и программы П, состоящей из последовательности команд π1,..,πn, компилятор однородно выбирает функцию f и множества Пf=(π1http://citforum.ru/security/articles/kazarin/plus.gif f(2k+1),2k+1),...,(πnhttp://citforum.ru/security/articles/kazarin/plus.gif f(2k+n),2k+n).

Так как общее время выполнения машины RAMk во всех экспериментах, инициированных противником, является не более 2k, мы никогда не используем тот же самый аргумент f для двух различных операций шифрования. Это следует из того, что шифрование (которое использует шифр "одноразовый блокнот") абсолютно безопасно (в информационно-теоретическом смысле), и следовательно, противник не получает никакой информации относительно оригинального содержания области значений.

Отметим, что на практике можно заменять случайный оракул на псевдослучайный. Следовательно, результат будет верен только для противников, ограниченных по времени некоторым полиномом. Компилятор на параметре входа k и программе П равномерно выбирает псевдослучайную функцию f. Описание f аппаратно реализовано в CPUk. Следовательно, ЦП способен вычислять f на входах длины k и poly(k)-временной противник может различать поведение этого ЦП от ЦП, описанного в доказательстве теоремы. Следовательно, любой противник, который может выполнять эксперименты по вмешательству за время, ограниченное poly(k), может быть вычислен за время, ограниченное poly(k), с доступом только к оракулу спецификаций. Подобные замечания верны и для следующей теоремы.

**Теорема 2.2.** Пусть {RAMk}k*http://citforum.ru/security/articles/kazarin/e.gif*N - вероятностная RAM-машина, которая выполняет забывающее моделирование (с метками времени) универсальной RAM-машины. Кроме того, предположим, что t шагов оригинальной RAM-машины моделируются меньше, чем за g(t)t шагов забывающей RAM-машины. Тогда существует компилятор, который защищает программное обеспечение от вмешивающихся противников, с затратами не более O(g(t)).

**Доказательство.** Для защиты от противника, который может изменять содержание сообщений, обмениваемых между ЦП и МП, используются схемы аутентификации. Без потери общности, ограничимся противником, который только изменяет сообщения в направлении от МП к ЦП.

Метка аутентификации будет зависеть от значения, которое хранится в фактической ячейке памяти и от числа предыдущих запомненных команд в этой ячейке. Интуитивно, такая метка аутентификации предотвращает возможность изменять значения, заменять его значением, хранимым в дру-гой ячейке, или заменять его значением, которое было сохранено ранее в той же самой ячейке.

Центральный процессор CPUk, рассмотренный в предыдущей теореме, далее модифицируется следующим образом. Модифицированная машина CPUk имеет доступ к еще одной случайной функции, обозначаемой f. Эта функция может быть объединена с другими. В случае, если CPUk должен сохранить зашифрованное значение v в ячейке памяти он сначала определяет текущий номер версии a. Отметим, что номер версии(a) может быть вычислен CPUk в соответствии с определением моделирования с метками времени. Модифицированная машина CPUk теперь посылает сообщение (сохранить, a, (v,f(a,версия(a),v))) вместо сообщения (сохранить ,a ,v), посланного первоначально. После получения сообщения (v,t) из МП в ответ на запрос (выборка ,a ,'), модифицированная машина CPUk определяет текущее значение номера версия(a) и сравнивает t с f(a, версия(a),v). В случае, если эти два значения равны CPUk работает как и прежде. В противном случае, CPUk немедленно останавливается, предотвращая, таким образом, защиту от вмешательства. Таким образом, попытки изменить сообщения от МП к ЦП будут обнаружены с очень высокой вероятностью.

Решение задачи "Квадратного корня"

Отметим, что тривиальное решение для забывающего моделирования RAM-машины заключается в полном сканировании фактической памяти RAMk-машины для каждой виртуальной операции доступа к памяти (которая должна быть выполнена для оригинальной RAM-машины). Далее описывается первое нетривиальное решение (принадлежащее О. Голдрайху) задачи забывающего моделирования RAMk-машины посредством вероятностной RAM'k'.

Пусть заранее известен объем памяти, обозначаемый m, требуемый для соответствующей программы. Ниже мы показываем, как моделировать такую RAM-машину посредством забывающей RAM-машиной с объемом памяти m+2http://citforum.ru/security/articles/kazarin/k.gif таким образом, что t шагов оригинальной RAM-машины моделируются за O(thttp://citforum.ru/security/articles/kazarin/k.gif) шагов на забывающей RAM-машине. Всякий раз, когда мы говорим о виртуальном доступе к памяти, мы подразумеваем доступ к памяти, требуемый для оригинальной RAM-машины, которая моделируется. Доступ к памяти при забывающем моделировании RAM-машины рассматривается как фактический доступ к памяти. Кроме того, без потери общности, будем понимать, что виртуальная операция доступа состоит из содержимого единственной ячейки памяти (т.е., выборка(i), сопровождаемая командами сохранить(i,') для некоторого i).

Общее описание алгоритма "Квадратного корня". Интуитивно, чтобы полностью скрыть виртуальную модель доступа, мы должны скрыть следующее:

* к каким виртуальным ячейкам осуществляется доступ и в каком порядке ?
* сколько раз к конкретной виртуальной ячейке осуществляется доступ (в случае, если к ней обращались)?

В первом случае достаточно каким-либо образом "перемешать" память так, чтобы противник не знал, какой фактический адрес памяти соответствует данному виртуальному адресу. Во втором случае, мы должны убедиться, что к любой ("перемешанной") локальной памяти осуществляется доступ более одного раза. Высокоуровневые шаги моделирования следующие.

**Инициализация:** Первые m+http://citforum.ru/security/articles/kazarin/k.gif слов RAM-моделирования расположены так, чтобы содержать m виртуальных адресов (к которым оригинальная RAM-машина обращается в процессе выполнения) и http://citforum.ru/security/articles/kazarin/k.gif"фиктивных" слов (см. рис.2.11). Оставшиеся http://citforum.ru/security/articles/kazarin/k.gif слов распределены так, чтобы обслужить дополнительную ("краткосрочную") память (здесь и далее называемую зщт).

**Моделирование шагов RAM-машины:**Пока моделирование RAM-машины не остановится, выполнить. (Моделирование выполняется за периоды, каждый из которых состоит из http://citforum.ru/security/articles/kazarin/k.gif шагов оригинальной/моделируемой машины). В каждом таком периоде выполняются следующие шаги.

Случайно переставить содержимое ячеек от 1 до m+*http://citforum.ru/security/articles/kazarin/k.gif*. То есть необходимо равномерно выбрать перестановку π над целыми числами от 1 до m+http://citforum.ru/security/articles/kazarin/k.gif и забывающим образом переместить содержимое виртуального слова i в фактическое слово πi. Необходимо подчеркнуть, что память зщт (т.е., ячейки от (m+http://citforum.ru/security/articles/kazarin/k.gif+1) до (m+2http://citforum.ru/security/articles/kazarin/k.gif)) не участвует в этом случайном перемешивании. Таким образом, фактические адреса от 1 до m+http://citforum.ru/security/articles/kazarin/k.gif называются перемешанной памятью.

Моделировать*http://citforum.ru/security/articles/kazarin/k.gif*виртуальных операций доступа к памяти оригинальной RAM-машины. В процессе моделирования мы сохраняем значения виртуальных операций доступа, восстановленных и модифицированных в течение текущего периода в памяти зщт. Так как размер памяти зщтравняется числу возможных операций доступа в одном периоде, можно хранить все значения, восстановленные в течение текущего периода в памяти зщт. Доступ к памяти оригинальной RAM-машины, скажем доступ к виртуальному слову i, моделируется следующим образом:

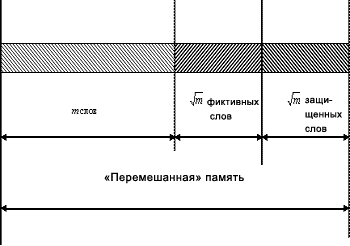


Рис. 2.11. Структура данных для решения задачи "квадратного корня"

* сканируется вся память зщт и проверяется, находится ли содержимое виртуального слова i в одном из слов памяти зщт. Подчеркнем, что здесь мы обращаемся к каждой ячейке памяти зщт в предопределенном порядке независимо от того, находится ли там виртуальное слово, которое нам надо;
* в случае, если i-тое виртуальное слово не найдено в памяти зщт, мы восстанавливаем его из фактического слова π(i), которое является текущей ячейкой i-того виртуального слова в течение этого периода;
* в противном случае (т.е., в случае, если i-тое виртуальное слово найдено в памяти зщт), мы получаем доступ к следующему "фиктивному слову" в перемешанной памяти (например, мы обращаемся к фактическому адресу π(m+j), где j - число шагов, моделируемых в текущем периоде);
* в любом случае модифицируемое значение для i-той виртуальной ячейки записано (забывающим образом) в память зщт посредством сканирования заново всех слов памяти зщт

.

**Модификация перемешанной памяти.** В конце периода, используются значения, сохраненные в памяти зщт для модификации забывающим образом содержимого перемешанной памяти.

Перед тем как приступить к деталям реализации вышеупомянутых шагов, сделаем несколько замечаний относительно того, почему они составляют забывающее моделирование. Далее покажем, как осуществить доступ к памяти на шаге 1 фиксированным образом а, следовательно, независимо от входа и виртуальной модели доступа оригинальной RAM-машины. Различают два типа операций доступ к памяти, выполненных на шаге 2: полное сканирование памяти зщт (т.е., осуществление доступа к каждому из слов дважды на каждую виртуальную операцию доступа) и осуществление доступа к http://citforum.ru/security/articles/kazarin/k.gif ячейкам перемешанной памяти во время каждого периода. Для каждых возможных http://citforum.ru/security/articles/kazarin/k.gif виртуальных операций доступа, последние http://citforum.ru/security/articles/kazarin/k.gif фактических операций доступа равномерно распределены среди всех http://citforum.ru/security/articles/kazarin/k.gifподмножеств {1,...,m+http://citforum.ru/security/articles/kazarin/k.gif}, где распределение вероятностей индуцировано выбором перестановки π. Таким образом, фактический доступ, выполняемый на шаге 2, не открывает никакой информации относительно виртуальных операций доступа, выполняемых в этом шаге. Легко увидеть, что шаг 3 не создает никаких новых трудностей, поскольку он может быть сделан при выполнении операций фактического доступа на шагах 1 и 2 в обратном порядке.

Реализация алгоритма "Квадратного корня"

Сначала, мы покажем, как выбирать и сохранять в ЦП случайную перестановку над {1,2,...,n}, используя O(log n) памяти и случайный оракул. Идея состоит в том, чтобы использовать оракул для проставления меток случайно выбранных элементов и различные целые числа из множества меток, обозначаемого Tn. Перестановка получается посредством сортировки элементов в соответствии с их метками. Если же необходимо получить вероятность коллизии ε (т.е., для нашего приложения ε=2-2k), достаточно иметь метки, выбранные случайно из множества Tn={1,2,...,n2/ε},. Пусть τ: {1,2,...,n}- > Tn - случайная функция, тривиально созданная случайным оракулом. В этом случае π(i)=k, тогда и только тогда, когда π (i) - наименьший элемент в {τ(j):1http://citforum.ru/security/articles/kazarin/b.gifj*http://citforum.ru/security/articles/kazarin/b.gif*n}. В нашем случае n=m+http://citforum.ru/security/articles/kazarin/k.gif , а именно n элементов состоят из m виртуальных адресов, связанных с целыми числами 1,...,m и http://citforum.ru/security/articles/kazarin/k.gif макетов, связанных с (m+1,..., m+http://citforum.ru/security/articles/kazarin/k.gif}.

Теперь мы имеем дело с задачей забывающей сортировки n элементов посредством меток. Определяющее условие состоит в том, что RAM-машина, которая выполняет сортировку, может хранить только фиксированное число значений одновременно. Идея состоит в том, чтобы "выполнить" сортирующую сеть Батчера, который позволяет сортировать n элементов, выполняя n|log2n| 2 сравнений. Каждое сравнение "выполняется" посредством осуществления доступа к двум соответствующим словам, чтением их содержания и записью этих значений обратно в необходимом порядке. Последовательность операций доступа к памяти, сгенерированной для этой цели фиксирована и не зависит от входа. Отметим, что забывающая RAM-машина может легко вычислять в каждой точке, какое сравнение требуется для реализации следующего. Это следует из простой структуры сети Батчера, которая является однородной относительно логарифмического пространства. Этот алгоритм будет работать, если мы сохраняем метку каждого элемента вместе с самим элементом (виртуальное слово или макет).

Далее мы точно определим, как осуществить доступ к виртуальной ячейке или к макету i. Отметим, что после шага 1 виртуальные ячейки от 1 до m (также как и макеты от m+1 до m+http://citforum.ru/security/articles/kazarin/k.gif) сортируются согласно их меток (т.е., τ(')). Таким образом, фактический доступ в перемешанную память на шаге 2 выполняется двоичным поиском необходимой метки. А именно, предположим, что мы хотим получить доступ к элементу ihttp://citforum.ru/security/articles/kazarin/e.gif {1,...,m+http://citforum.ru/security/articles/kazarin/e.gif}. Затем, вместо того, чтобы непосредственно достичь фактической ячейки π(i), как предлагается выше, мы выполняем двоичный поиск метки π(i). Этот двоичный поиск заканчивается на фактической ячейке π(i). Помимо этого другие фактические операции доступа, выполняемые во время поиска, полностью определены π(i). Таким образом, эти дополнительные фактические операции доступа не открывают никакой информации наблюдателю (противнику).

Далее описываются две альтернативных реализации шага 3. Первый вариант - реверсия модели доступа на шаге 2. Второй вариант - полная сортировка фактической памяти (то есть, все m+2http://citforum.ru/security/articles/kazarin/k.gif слов, включая память зщт) дважды как описано в алгоритме. Первая сортировка выполняется в соответствии с ключом (v,σ ), где v - виртуальный адрес (http://citforum.ru/security/articles/kazarin/vv.gif - для макетов) и σhttp://citforum.ru/security/articles/kazarin/e.gif {0,1} указывает, исходит ли это слово из памяти зщт или из перемешанной памяти. Таким образом, сортируемый список имеет виртуальные адреса, появляющиеся так, чтобы некоторые из них появляются в двойном экземпляре, один за другим (одна версия из памяти зщт, а другая из перемешанной памяти). Затем, мы сканируем этот список и для каждого виртуального адреса, появляющегося в дубликате, маркируется второе местонахождение (возникающее из перемешанной памяти) также как и макет (т.е., http://citforum.ru/security/articles/kazarin/vv.gif ). В заключение, мы снова полностью сортируем память, но на сей раз виртуальными адресами. Фактически, эта дополнительная сортировка не нужна, поскольку память будет подвергнута пересортировке когда-либо при следующем выполнении шага 1.

Далее следует детальное описание шага 2. Главная идея при этом моделировании состоит в том, чтобы осуществить доступ к каждой виртуальной ячейке в "перемешанной памяти" только в течение каждого шага периода. Как только "осуществиться" доступ к некоторой виртуальной ячейке, необходимо сохранить версию этой виртуальной ячейке в памяти зщт. В течение шага 2, счт содержит число виртуальных операций доступа, моделируемых в текущем периоде. Переменная счт - первоначально содержит 0 и увеличивается, пока достигнет значения http://citforum.ru/security/articles/kazarin/k.gif . Булева переменнаяfoundбудет инициироваться, если требуемое значение найдено в памяти зщт. Когда оригинальная RAM-машина осуществляет доступ к i-тому виртуальному слову, забывающая RAM-машина работает следующим образом:

* (2a) сканирует полностью память зщт и ищет виртуальный адрес i. А именно, для j, пробегающему значения от m+http://citforum.ru/security/articles/kazarin/k.gif +1 до m+2http://citforum.ru/security/articles/kazarin/k.gif, доступ к фактической ячейке памяти j переменная foundустанавливается в true и сохраняется в ЦП, если виртуальный адрес i совпадает с фактической ячейкой j. (Переменнаяfound инициализирована в значение false до этого сканирования и остается такой же, если виртуальный адрес i не был найден);
* (2b) если found=false, тогда забывающая RAM-машина осуществляет доступ к слову с меткой π(i) и сохраняет содержимое в ЦП. Как показано выше, это реализуется посредство двоичного поиска метки π(i);
* (2c) если found=true, тогда забывающая RAM-машина осуществляет доступ к слову с меткойπ(m+счт), которое является "макетом". Это также реализуется посредством двоичного поиска меткиπ(m+счт);
* (2d) просматривает полностью память зщт снова и записывает модифицируемое значение i-того виртуального слова в памяти зщт. А именно, для m+http://citforum.ru/security/articles/kazarin/k.gif +1 до m+2http://citforum.ru/security/articles/kazarin/k.gif доступ к фактической ячейке памяти j запоминается в ее модифицированном значении виртуального адреса i, если адрес jсодержит старое значение виртуального адреса i (т.е., found=true), либо found=false и j - первое пустое слово в shelter-е.
* Значение счт увеличивается на 1.

Подчеркнем, что наблюдатель не может видеть, сохранил ли ЦП значения или нет и, таким образом, не может различать выполнение шага 2b от выполнения шага 2c. Ясно, что шаги 2a и 2d имеют фиксированную модель доступа и, таким образом, не никакая информация, полезная для наблюдателя, не вскрывается.

Анализ алгоритма "Квадратного корня"

Как обсуждалось выше, последовательность фактических операций доступа к памяти забывающей RAM-машины действительно не открывает никакой информации относительно последовательности виртуальных операций доступа к памяти оригинальной RAM-машины. Это действительно так, потому что во время шагов 1, 2a, 2d и 3 фактическая модель доступа фиксирована, в то время как во время шагов 2b и 2c фактические модели доступа неразличимы и "случайны".

Теперь остается вычислить затраты на моделирование (т.е., отношение числа операций доступа, выполненных на забывающей RAM-машиной к числу оригинальных операций доступа). Далее мы вычисляем общее число фактических операций доступа, выполняемых на период (т.е., m+2http://citforum.ru/security/articles/kazarin/k.gifвиртуальных операций доступа). Число фактических операций доступа на шаге 1 определено числом сравнений в сортирующей сети Батчера, а именно, O(mlog2m). То же самое делается на шаге 3. Что касается шага 2, каждая виртуальная операция доступа выполняется за 2http://citforum.ru/security/articles/kazarin/k.gif +log2(m+http://citforum.ru/security/articles/kazarin/k.gif)=O(http://citforum.ru/security/articles/kazarin/k.gif) фактических операций доступа. Это составляет амортизационные затраты O(log2mhttp://citforum.ru/security/articles/kazarin/k.gif). Фактически, вышеупомянутый выбор параметров (то есть, размер памяти зщт) не оптимален.

При использовании памяти зщтразмера s, мы получаем амортизационные затраты

http://citforum.ru/security/articles/kazarin/51.gif

которые минимизированы установкой s=Θ(logmhttp://citforum.ru/security/articles/kazarin/k.gif ).

Заключительные замечания

В данном подразделе был представлен компилятор, который трансформировал RAM-программы в эквивалентные программы, которые предотвращают попытки противника выяснить что-либо относительно этих программ при их выполнении. Перенос был выполнен на уровне команд, а именно операции доступа к памяти для каждой команды заменялись последовательностью избыточных операций доступа к памяти. Понятно, что все формулировки и результаты, показанные выше, применимы к любому другому уровню детализации выполнения программ. Например, на уровне "пролистывания" памяти это означало бы, что мы имеем дело с операциями "получить страницу" и "сохранить страницу", как с атомарными (базовыми) командами доступа. Таким образом, единственная операция "доступ к странице" заменяется последовательностью избыточных операций "доступ к странице". В целом исследуется механизм для забывающего доступа к большому количеству незащищенных ячеек памяти при использовании ограниченного защищенного участка памяти. Применение к защите программ было единственным приложением, обсужденным выше, но возможны также и другие приложения.

Одно возможное приложение - это управление распределенными базами данных в сети доверенных сайтов, связанных небезопасными каналами. Например, если в сети сайтов нет ни одного, который содержал бы полную базу данных, значит необходимо распределить всю база данных среди этих сайтов. Пользователи соединяются с сайтами так, чтобы можно было восстановить информацию из базы данных таким образом, чтобы не позволить противнику (который контролирует каналы) изучить какая часть базы данных является наиболее используемой или, вообще, узнать модель доступа любого пользователя. В данном случае не требуется скрывать факт, что запрос к базе данных был выполнен некоторым сайтом в некоторое время, - просто надо скрывать любую информацию в отношении фрагмента необходимых данных. Также принимается предположение о том, что запросы пользователей выполняются "один за другим", а не параллельно. Легко увидеть, что забывающее моделирование RAM-машины может применяться к этому приложению посредством ассоциирования сайтов с ячейками памяти. Роль центрального процессора будет играть сайт, который в текущий момент времени запрашивает данные из базы и информация о моделировании может циркулировать между сайтами забывающим способом. Отметим, что вышеупомянутое приложение отличается из традиционной задачи анализа трафика.

Другое приложение - это задача контроль структуры данных, которая следует из определений самотестирующихся программ, рассматриваемых выше. В этой конструкции желательно сохранить структуру данных при использовании малого количества достоверной памяти. Большая часть структуры данных может сохраняться в незащищенной памяти, где и надо решать задачу защиты от вмешательства противника. Цель состоит в том, как обеспечить механизм контроля целостности данных, которые необходимо сохранять посредством забывающего моделирования RAM-машиной.